Universidade do Minho

Scalable adaptive collaborative filtering

João Nuno Vinagre Marques da Silva

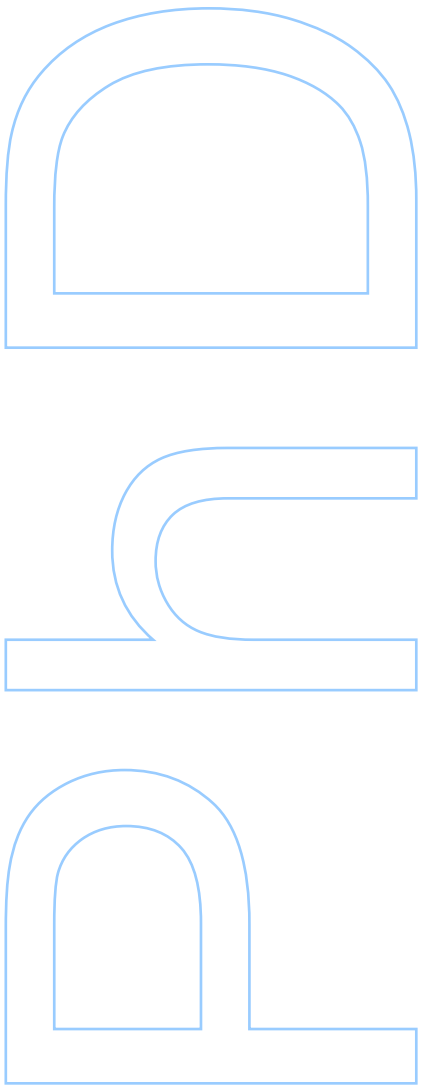
Tese de Doutoramento apresentada à
Faculdade de Ciências da Universidade do Porto
Informática
2016

Scalable adaptive collaborative filtering

João Nuno Vinagre Marques da Silva
MAPi - Doutoramento em Informática
Universidade do Porto - Faculdade de Ciências
Departamento de Ciência de Computadores
LIAAD - INESC TEC
2016

Orientador
Alípio Mário Guedes Jorge
Professor Associado
Faculdade de Ciências da Universidade do Porto
LIAAD - INESC TEC

Coorientador
João Portela da Gama
Professor Associado
Faculdade de Economia da Universidade do Porto
LIAAD - INESC TEC



Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Alípio Jorge and my co-supervisor, Prof. João Gama. Their commitment to my success in the development of this thesis was notorious, both by the enriching discussions we had and by their constant encouragement to pursue my goals. Also, of course, for their friendship.

During more than four years, I had the chance to share my work experiences with colleagues from the Computer Science Department in the Faculty of Sciences of the University of Porto and LIAAD-INESC TEC. I particularly thank Amir Nabizadeh with whom I shared the workplace and had the most fun discussions about recommender systems. I also thank everyone at LIAAD for their friendship and the great work environment. A huge thanks to Joana Dumas, for always being there for me (and everyone else) with the traveling, scheduling, form filling, and all the bureaucracy.

I was fortunate to have met Pawel Matuszyk, with whom I have collaborated, together with our supervisors. We became friends. Like me, he is a big enthusiast of recommender systems and we share many thoughts on science, technology, R&D and other related subjects. Our discussions on recommender systems were tremendously helpful for my work and, I hope, for his work too.

My work could not have been conducted without the financial support by FCT - Fundação para a Ciência e Tecnologia, the portuguese foundation for science and technology, with the research grant SFRH / BD / 77573 / 2011. I am also grateful for the funding from European Regional Development Fund (ERDF) through FCT in projects “SIBILA” (NORTE - 07 - 0124 - FEDER - 000059) and “PEST” (FCOMP - 01 - 0124 - FEDER - 037281) and for the European Commission’s support of project MAESTRA (Grant no. ICT - 2013 - 612944). Part of the work on this thesis is also integrated in Project “TEC4Growth” (NORTE - 01 - 0145 - FEDER - 000020), financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

Finally, I would like to thank my family. I thank my parents for supporting me unconditionally and for being proud of me. My greatest thank is for the two most important and beloved women in my life, my girlfriend Eliana and my daughter Miriam. For putting up with me and for giving meaning to everything I do. This work would not have taken place, and this thesis would not exist without them. Obrigado!

Abstract

Online communities are growing. Human beings, both as individuals and as organizations, benefit from the availability of easily accessible information. This contributes to the dissemination of information, knowledge and culture. However, it also makes information filtering harder, due to the overwhelming amount of content from a huge variety of sources. Search engines play a fundamental role in helping users to find relevant content. However, they have the limitation that users need to know what they are looking for, and how to describe it in a query string.

Recommender systems complement search engines by recommending relevant content to users, even if they are not aware of the existence of that very content. This is especially helpful when choosing from a large collection of items, such as in music streaming services, TV on demand providers, book stores or travel agencies. Collaborative Filtering is a powerful technique to compute personalized recommendations. It uses preference history from a large community of users to infer personalized preferences for individual users.

In most real-world systems, recommender algorithms work in an environment where data is continuously being generated. In this thesis, we address the problem of learning recommendation models from such continuous flows of user-generated data – data streams. We focus specifically on streams of positive-only data, that consists exclusively of user-item interactions indicating positive preferences. We propose algorithms that are able to deal with the streaming data environment, as well as learn accurate recommendation models from positive-only user feedback data.

We compare our algorithms with both classic and state-of-the-art alternatives. To do this, we use a highly informative evaluation methodology – prequential evaluation – specifically designed for algorithms that learn from data streams. Our results show that our proposals are able to outperform the alternatives in terms of both predictive ability and running time.

Resumo

As comunidades online estão a crescer. O ser humano, individualmente ou em organizações, beneficia da disponibilidade de informação facilmente acessível. Isto contribui para a disseminação de informação, conhecimento e cultura. No entanto, dificulta a filtragem da informação, dada a esmagadora quantidade de conteúdos e a enorme variedade de fontes. Os motores de pesquisa desempenham um papel fundamental, ajudando utilizadores a encontrar conteúdos relevantes. No entanto, exigem que eles saibam exatamente o que procuram, e como descrevê-lo textualmente.

Os sistemas de recomendação complementam os motores de pesquisa encontrando conteúdos relevantes para os utilizadores, mesmo que estes não tenham conhecimento prévio da sua existência. Isto é especialmente útil na escolha entre uma grande coleção de itens, como acontece em serviços de *streaming* de música, *TV on demand*, livrarias ou agências de viagem. A Filtragem Colaborativa é uma técnica poderosa que produz recomendações personalizadas. Usa o histórico de preferências de uma grande comunidade de utilizadores para inferir preferências para utilizadores individuais.

Na maioria dos sistemas reais, os algoritmos de recomendação funcionam num ambiente em que os dados são continuamente gerados. Nesta tese, abordamos o problema da aprendizagem de modelos de recomendação a partir desses fluxos contínuos de dados. Focamos especificamente em dados só-positivos, que consistem em interações utilzador--item que indicam exclusivamente preferências positivas. Propomos algoritmos capazes de lidar com o ambiente the fluxo contínuo de dados, bem como aprender modelos de recomendação com alta precisão a partir de dados só-positivos.

Comparamos os nossos algoritmos com alternativas do estado-da-arte e clássicas. Para fazê-lo, usamos uma metodologia de avaliação altamente informativa – avaliação prequencial – especificamente desenhada para algoritmos que aprendem a partir de fluxos de dados. Os resultados obtidos mostram que as nossas propostas são capazes de superar as alternativas em termos de capacidade preditiva e tempo de operação.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The amount of content available on the internet, as well as in enterprise and governmental information systems is overwhelming. It is not possible for human beings to browse through all the content available on large online commerce catalogs, text-based and multimedia databases and social networks, without some kind of automated filtering. This has long ago motivated the development of information filtering algorithms. The most obvious example are implemented in search engines such as Yahoo!, Google or Bing. These algorithms typically allow users to conveniently filter content using simple text queries. For example, if a user wants to find a strawberry pie recipe, she would naturally query her favorite search engine with the text "strawberry pie recipe". The algorithm then matches the query to a *huge* collection of indexed content and displays the results as a list, usually ordered by relevance.

The importance of information filtering is nowadays evident. For instance, in December 2014, 3 of the top 5 most active internet domain names belong to general purpose search engines [Alexa Website Rankings, 2015; Similarweb Website Rankings, 2015], according to Alexa[1] and SimilarWeb[2], two popular worldwide traffic rankings. However search engines are designed for on-demand retrieval, which means that users have to know what they are looking for. This is not always possible or even desirable. When a user wishes to find content that is unknown to her, she obviously would not know how to formulate a text query to describe it. Typical examples of this are the discovery of music, movies, books, TV shows, touristic destinations, news and e-learning content, although there is no fundamental restriction on the type of content to be discovered. For instance, when a user wishes to find new movies to watch or music to listen to, the problem is how to find *novel* content that matches the user's preferences. This is one example of a typical recommendation task, and systems that aim to solve this and other similar problems are

[1] http://www.alexa.com
[2] http://www.similarweb.com

known as *recommender systems*.

Over the past two decades, an increasing number of researchers has focused on Recommender Systems. Collaborative Filtering (CF) is one extensively studied recommendation method. It consists of using historical information about all users in a system to perform personalized recommendations to specific users. CF has been successfully used in a large number of applications, such as e-commerce websites [Linden et al., 2003] and other on-line communities in a series of domains [Resnick et al., 1994; Hill et al., 1995; Shardanand and Maes, 1995]. Competitions like the Netflix prize [Bennett et al., 2007], the KDD-Cup 2011 [Dror et al., 2012], the Million Song Challenge [McFee et al., 2012], the successive ACM RecSys Conferences, Workshops and Challenges, as well as the increased demand from both the academic community and the industry, have motivated many notable contributions in the field.

1.1 Motivation

Real-world online systems continuously generate data. However, most algorithms and techniques available in the literature on recommender systems fail to acknowledge this. Instead, they are designed to learn recommendation models from large static datasets containing user feedback collected from the system in which they operate. The dataset is then analyzed and a model is inferred. This model then remains essentially unchanged until another large chunk of data is available to retrain a new model. This approach raises two problems. The first problem arises from the system's unawareness of the continuous activity of users in the system for arbitrary amounts of time. The exact same model is used to perform recommendations for minutes, hours, days, weeks, months or years. As user preferences change over time and users and items enter and leave the system, the model becomes increasingly inaccurate. The second problem is computational. As the amount of collected data increases, the necessary computational resources to store it and process it also increase. Processing ever-growing data in batch eventually leads to scalability issues, even if the algorithm complexity grows linearly with the size of data. Systems either need to be scaled up – which can be expensive – or data needs to be reduced – which can mean throwing away potentially valuable information.

Many researchers have focused on both of these problems, but mostly independently of each other. Time related issues in recommender systems have been investigated in many contributions in the field [Campos et al., 2014; Vinagre et al., 2015b]. Regarding the computational problem, algorithms are becoming increasingly efficient in learning from large amounts of data. Additionally, distributed models are becoming more mature, flexible and widespread, enabling the use of recommendation algorithms and techniques that otherwise

would not be applicable [Low et al., 2012; Owen et al., 2011]. However, even this paradigm still has the fundamental limitation that the volume of data eventually becomes too high to be stored and processed.

One possible alternative is to consider data stream approaches [Domingos and Hulten, 2000] and apply them to the recommendation problem [Vinagre et al., 2014b; Matuszyk and Spiliopoulou, 2014]. Algorithms that learn from online data streams acknowledge two important aspects. First, the concepts that algorithms try to capture are typically non-stationary – they vary with time. In this sense, these concepts constitute a moving target. Second, data streams are potentially unbounded, while computational resources are limited and/or expensive. Ideally, algorithms should be able to operate independently from the number of examples. By approaching user feedback data that recommenders learn from as a data stream, these two aspects are inherently acknowledged. Algorithms that deal with data streams have been successfully used in many fields of application, such as medical systems, energy systems and computer networks [de Andrade Silva et al., 2013; Aggarwal, 2014]. However, recommender systems do not typically approach data as a stream.

In this thesis, we approach recommendation as a data stream problem, simply by acknowledging that the process that generates user feedback data – used in the training of recommendation models – is continuous and never stopping. We use algorithms that are able to learn from such streams of data by maintaining incremental models, and evaluate them using well studied protocols designed for streaming environments.

In this thesis we focus exclusively on positive-only user feedback streams. These are streams that only contain positive interactions between users and items, and provide no information on non-positive – neutral and negative – opinions. The main motivation to focus our research on positive-only data is availability. Practically any system that involves users and items already has a source of positive-only feedback. The examples are countless: web logs, music streaming history, shopping habits, social network sharing and "liking", news reading, watched movies or series, point-of-interest check-ins, to name just a few of them. These typically consist of continuous flows of user-item pairs, that indicate a positive preference of a user for an item. Unlike recommender systems that deal with ratings – like a numeric scale, like 1 to 5 stars – systems that deal with positive-only data face the problem of not having information about negative preferences – the items that users do not like. This hardens the task of finding good items to recommend, because it is not easy to distinguish between items users do not like – not good for recommendation –, and items that users simply do not know – the ideal recommendation candidates [Vinagre et al., 2015a; Pan et al., 2008]. Additionally, algorithms that learn from positive-only data are also usable with ratings data, which makes them universally applicable to user-item interaction datasets.

Given our general problem setting, which is to learn recommendation models from positive-

only user feedback streams, we also need to use an evaluation methodology that is applicable to this setting. Traditional methods designed for batch-learning algorithms are not suitable for algorithms that learn from data streams [Gama et al., 2013; Vinagre et al., 2014a], so we need to use a framework that allows the continuous evaluation of such algorithms. Evaluation methods and protocols for data stream algorithms are available in the literature, however we are not aware of any research that uses those methods in recommender systems. The evaluation of recommendation algorithms is typically done in offline, controlled environments. When compared to recommender systems running online with real users, in such a laboratory environment, researchers have more independence and freedom, in the sense that they do not depend so much on third parties. By avoiding the constraints of real-world applications, it is possible to focus research on very specific problems, without having to worry about external factors. However, as a natural consequence, problems caused by the very constraints that are circumvented in offline research are very rarely addressed. To evaluate our work, we use an evaluation methodology specifically designed for algorithms that learn from data streams that is designed to be used in online environments.

1.2 Research questions

The motivation above leads to the following four research questions (RQ).

RQ1 **Do phenomena related with time have a significant impact on recommendation? If so, is the existing knowledge in the field of recommendation sufficient to approach time related problems?** These phenomena encompass user preference changes over time – which can be fast and abrupt or slow and gradual –, the continuous incoming and outgoing of new users and items that naturally happen in online systems, seasonal effects. To answer this question, we study the state-of-the-art of usage-based recommendation techniques that deal with time, explicitly or implicitly. We address this question in Chapter 2.

RQ2 **Are batch learning approaches adequate and sufficient to deal with all the challenges of real world recommender systems? Could the techniques and algorithms for data streams be used to improve the accuracy and/or scalability of recommender systems?** To answer this, we argue that the ever increasing amount of data necessarily leads to both accuracy loss and heavy constraints on performing batch learning, given that computational resources are always limited. Based on the available knowledge on algorithms that learn from data streams, we investigate novel methods that are able to incrementally maintain recommendation models, as well as techniques to exploit the time dimension in the context of recommendation.

We propose an incremental matrix factorization algorithm based on state-of-the-art methods, designed to deal with incoming data streams of user feedback. We assess the benefits of using such techniques under the typical constraints of data stream environments. This question is addressed in Chapters 3 and 4.

RQ3 **Considering the known problems of dealing with positive-only data for recommendation, can we devise a scheme that mitigates those problems that is compatible with the streaming approach? Can we exploit the time dimension to do this?** Given that we use positive-only data, we need to deal with the effect of not having negative feedback to help the algorithms in distinguishing between good and bad recommendations. We propose a temporal scheme based that uses the recency of occurrence of items to select negative examples that are artificially introduced in the stream of feedback data. We address this in Chapter 4.

RQ4 **Are traditional evaluation protocols suitable for recommender systems running on dynamic real-world environments? If not, how de we evaluate recommendation algorithms in such environments?** In order to evaluate algorithms, we need an evaluation methodology that enables fair and accurate comparison between algorithms that maintain dynamic models. We use an evaluation framework based on well studied evaluation methods for data stream mining. We use this framework to assess the accuracy and speed of our proposed algorithms. We cover this in Chapter 5.

1.3 Contributions

In this thesis we provide four contributions.

- We survey the state-of-the-art on recommendation systems, with particular focus on the ones that deal with temporal effects. Based on this study, we find that exploiting time is generally beneficial in recommender systems, and identify the most important related challenges.
- We propose a matrix factorization algorithm that learns a recommendation model in fast incremental steps. This enables the model to continuously learn from single data points as they arrive in a stream of user feedback.
- We show that diversity techniques are beneficial in recommendation models that learn from data streams. We propose the use of online bagging – bootsrap aggregating – to improve the accuracy of our incremental matrix factorization algorithm.

- We introduce two recency based mechanisms to (a) solve the problem of learning exclusively from positive examples and (b) exploit the recency of occurrence of items, taking advantage of the time dimension.
- We devise an evaluation methodology specifically for recommender systems that learn from data streams. Typical evaluation protocols used with batch algorithms are simply not applicable in an environment where data keeps flowing in. We use a well known evaluation framework for data streams – prequential evaluation – to evaluate recommendation algorithms in a streaming environment. We also present new forms of visualization of results, as well as the visualization of statistical significance of the difference between algorithms. These visualizations show the evolution of performance of algorithms over time, as they process incoming data.

1.4 Organization

After this chapter, this thesis is divided in five more chapters.

In Chapter 2, we discuss the concepts that constitute the building blocks in our research, focusing on RQ1. We identify the tasks of recommender systems, and provide a general view on the classes of algorithms that can be used to accomplish those tasks, with special focus on the ones that are directly useful for the understanding of the remainder of the thesis. We also provide an overview on algorithms that exploit time information to improve recommendation.

Chapter 3 focuses on RQ2, specifically on recommendation algorithms that are capable of learning from data streams of usage data. We start by describing the specific properties data streams, and the requirements of algorithms that learn from such flows of data. We analyze several algorithms and techniques that are suitable for recommendation with streaming data.

To answer the second part of RQ2, we propose, in Chapter 4, an incremental matrix factorization algorithm (Algorithm 4.2 - ISGD), along with BaggedISGD, an ensemble version of ISGD that uses bagging, and also RAISGD and RAISGD-RB, two versions of ISGD that use recency-based technique to tackle the challenges of learning from positive-only (specifically addressing RQ3).

In Chapter 5 we evaluate our proposed algorithms, essentially addressing RQ4. We start by describing the prequential evaluation process, as used in data streams. Then we use that methodology to evaluate our proposals, in four stages. First, we measure the benefits of using ISGD by comparing it with its batch-learning version. Second, we evaluate the use of bagging with ISGD. Third, we evaluate the recency-based negative feedback imputation

schemes presented in Chapter 4, by comparing how several amounts of negative feedback perform with respect to the original ISGD algorithm (without negative feedback imputation). Fourth, we compare ISGD, with and without negative feedback imputation, with a classic and well known neighborhood-based algorithm and two versions of a state-of-the-art algorithm.

Finally, we draw conclusions in Chapter 6, by presenting the core of our findings, the limitations of our proposals, and future lines of work that can complete or complement this thesis.

We also add three Appendixes. We list abbreviations used in the text of the thesis in Appendix A. To avoid the excessive proliferation of graphics in Chapter 5, we move to Appendix B several auxiliary plots. These plots are referenced in the text where they may become relevant.

1.5 List of publications

The following papers have been published (by order of appearance):

- [Vinagre et al., 2014b] João Vinagre, Alípio Mário Jorge, and João Gama. *Fast incremental matrix factorization for recommendation with positive-only feedback*. In Proceedings of the 22nd International Conference on User Modeling, Adaptation, and Personalization (UMAP 2014), Aalborg, Denmark, July 7-11, 2014. Volume 8538 of Lecture Notes in Computer Science, pages 459–470. Springer, 2014.
- [Félix et al., 2014] Catarina Félix, Carlos Soares, Alípio Mário Jorge, and João Vinagre. *Monitoring recommender systems: A business intelligence approach*. In Proceedings of the 14th International Conference on Computational Science and Its Applications (ICCSA 2014), Guimarães, Portugal, June 30 - July 3,2014. Volume 8584 of Lecture Notes in Computer Science, pages 277–288. Springer, 2014.
- [Vinagre et al., 2014a] João Vinagre, Alípio Mário Jorge, and João Gama. *Evaluation of recommender systems in streaming environments*. In Proceedings of the Workshop on Recommender Systems Evaluation: Dimensions and Design (REDD 2014) in conjunction with the 8th ACM Conference on Recommender Systems (RecSys 2014), Foster City, CA, USA, October 10, 2014.
- [Vinagre et al., 2015a] João Vinagre, Alípio Mário Jorge, and João Gama. *Collaborative filtering with recency-based negative feedback*. In Proceedings of the 30th ACM SIGAPP Symposium on Applied Computing (SAC 2015), April 13-17, 2015, Salamanca, Spain. Pages 963–965. ACM, 2015.

- [Matuszyk et al., 2015] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. *Forgetting methods for incremental matrix factorization in recommender systems*. In Proceedings of the 30th ACM SIGAPP Symposium on Applied Computing (SAC 2015), April 13-17, 2015, Salamanca, Spain. Pages 947–953. ACM, 2015.
- [Vinagre et al., 2015b] João Vinagre, Alípio Mário Jorge, and João Gama. *An overview on the exploitation of time in collaborative filtering*. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 5(5):195–215, 2015.

In [Vinagre et al., 2014b], we present two contributions directly related to this thesis. First, we propose the incremental matrix factorization algorithm for positive-only data described in Chapter 4 (Algorithm 4.1). We also introduce the evaluation methodology described with detail in Chapter 5. In [Vinagre et al., 2014a], we focus specifically on the benefits of this evaluation methodology, namely by using significance tests over time.

In [Vinagre et al., 2015a], we propose a recency-based scheme that improves the performance of algorithms that deal with positive-only data. This contribution is also directly related to this thesis, and is described in Section 4.3.

A survey on the exploitation of the time dimension is available in [Vinagre et al., 2015b]. This is a contribution that focuses on algorithms that are able to take advantage of the time dimension to improve recommendation. This contribution is also available in Section 2.6.

Finally we had two collaboration works. In [Matuszyk et al., 2015], we study forgetting strategies for recommendation with incremental matrix factorization algorithms. This work is closely related to our past work in [Vinagre and Jorge, 2012], and is a parallel line of work of this thesis. We describe our major findings in Section 3.4. In [Félix et al., 2014], we study the performance monitoring of recommender systems from a business intelligence perspective. Our findings show that business intelligence tools are effective in monitoring the performance of recommendation models over time.

Chapter 2

Recommender Systems

This thesis focuses on specific issues of recommender systems. However, to describe these issues, some introductory background and context are necessary. This chapter provides a general introduction to recommender systems, focusing on the concepts, definitions and techniques that constitute the framework for the remainder of the thesis. In Section 2.6, we also provide an overview on the emergent techniques that deal with time-related aspects, such as seasonality or preference changes over time.

2.1 Tasks of a recommender system

Recommender systems are typically designed for online communities encompassing a large number of users – typically human beings – that browse through a large number of items in the system. Items can be movies, music, books, touristic attractions, restaurants or any other kind of product or content of interest. Users and items are the two central entities in all recommendation problems.

Even though the idea behind all recommender systems is essentially the same – recommend items to users –, there is quite a variety of end-user tasks in which recommender systems can assist. Herlocker et al. identify the following tasks in [Herlocker et al., 2004]:

- *Annotation in context*: this consists on the annotation of items with a value that indicates how much will the item be *liked* by the user. The recommender system only needs to predict this value. The most obvious example is predicting the rating – e.g. in a one to 5 star scale – a user would give to any particular item, and displaying that to the user. This is also known as *rating estimation* or *rating prediction*.
- *Find good items*: many users use recommender systems to find a set of items that

consists of the best items for them, i.e. the items the user is most likely to prefer. Recommender systems can achieve this either by sorting the top-N items according to a score – e.g. their predicted rating –, or by directly approaching the recommendation task as a ranking problem. This task is often referred to as *top-N recommendation*.

- *Find all good items*: here the task is similar to *Find good items* task, with the requirement of not missing any good recommendation.
- *Recommend sequence*: when items are consumed sequentially, the order by which they are recommended is crucial. Automatic music playlist generation is one good example of this task. The main objective is to find a good *sequence* of items, rather than a set – most likely ordered by relevance – that the user can access in an arbitrary order.
- *Just browsing*: some users may just find interesting or entertaining to browse through a provider's catalog, even if there is no imminent interest of consumption and/or purchase. Users here are more interested in the navigation functionality and style provided by recommender systems.
- *Find credible recommender*: some end-users browse through several recommender systems just testing how well they match their actual tastes, until they find one that is satisfactory. This is a common task of new users. Of course, while testing, users can only evaluate recommendations of items that they already know, so there is a natural bias towards perhaps less valuable recommendations.

All of the above tasks have implications in the problem formulation and evaluation methodology and metrics. For example, the metrics used to evaluate *rating estimation* are naturally different from the ones used for evaluating the recommendation of item *sequences*. The measurement of the overall quality of a recommender system is highly dependent on the task for which the recommender is designed.

As Herlocker et al. state in [Herlocker et al., 2004], the vast majority of research focuses on either one of the two first tasks. In this thesis we focus on *top-N recommendation – Find good items*.

2.2 Content-based filtering

Content-based filtering provides recommendations by analyzing user profiles or history [Pazzani and Billsus, 2007], and matching the obtained information to the items' content. For example, in a system that recommends news, if the available information about a user

indicates a preference for technology and arts, then news with relevant content about these fields of interest will be recommended. This approach requires that some information about the user is available. This information can be obtained in two ways [Pazzani and Billsus, 1997; Mooney and Roy, 2000]:

- *explicitly*: user profiles are built based on information, such as demographic information or preferences explicitly specified, provided by the users themselves;
- *implicitly*: user activity history, such as purchased items on an e-commerce website, is analyzed and recommendations are made accordingly.

Content-based filtering works when information about users is available and when it is possible to extract information from the items content that we can relate to user profiles or history. The first problem arises from the need to obtain information about users, since it is not possible to make reliable recommendations if user information is not available or incomplete [Adomavicius and Tuzhilin, 2005].

Another problem consists of the limitations of content analysis [Shardanand and Maes, 1995; Balabanovic, 1997]. It is a fact that computers have limited abilities when it comes to interpret and recognize content, especially non-textual content. While text content is relatively easy to analyze and extract information from, analyzing multimedia content – for example, audio, video or images – can potentially become an extremely complex task. For instance, recommending action movies to a user known to like them would require that the system was able to accurately detect genre in video files. The most obvious and used way to work around this problem is to add meta-data to items, such as text attributes [Pazzani and Billsus, 2007]. This way it is possible to relate items to user preferences. In many systems, however, the large number of items can make it prohibitive to add attributes to items [Shardanand and Maes, 1995].

A system based on user profiles may also suffer from super-specialization [Balabanovic, 1997]. When a user profile determines some preference, only items that match that preference are recommended. In many cases, this can be the desired behavior, but in other cases it can be a limitation. For example, a user with a preference for italian restaurants would never receive a recommendation for other type of restaurants, no matter its popularity or quality. On the other hand, recommending all italian restaurants is probably not a useful recommendation.

Although not abundant in recent literature – when compared with collaborative filtering –, recent advances in multimedia content analysis, such as image and speech recognition, have provided new tools and techniques for the research in content-base recommendation. Techniques based on Neural Networks combined with optimization based learning, frequently referred to as Deep Learning [Bengio, 2009; Arel et al., 2010; Schmidhuber, 2015],

together with the advances in parallel computation have allowed the large scale analysis of multimedia content. This research is, however, out of the scope of this thesis.

2.3 Collaborative filtering

As mentioned in the beginning of this chapter, many on-line virtual communities consist of a large number of users that browse through items in the system. Items can be movies, music, books, touristic attractions, restaurants or any other kind of product of interest. In such systems, users are frequently allowed to give their personal opinion about items, by rating that item either explicitly – e.g. using a "like" button or a 5 star rating scale – or implicitly – e.g. number of times a user listens to a music track, or whether a user has bought some item or not. Suppose a system has n users and m items. By collecting feedback from users, it is possible to build a user-item feedback matrix $R_{n \times m}$ containing all ratings given by users to items. Typically R is a very sparse matrix – users usually only rate a very small proportion of the items in the system. By exploiting the preferences of similar users, Collaborative Filtering (CF) algorithms try to make predictions about individual user preferences. CF algorithms achieve this by *learning* or *training* a predictive model with the available usage data: the matrix R. This model can then be used to *predict* the user preferences that are missing in R. The best way to train this recommendation model is dependent on the type of feedback being analyzed.

2.3.1 Types of feedback

One important distinction, which determines the choice or development of an algorithm, is the one between the two possible types of user feedback data:

- Numeric ratings feedback: typically composed of triples in the form (u, i, r), consisting of the rating value r being given by user u to item i;
- Positive-only or unary feedback: a set of pairs in the form (u, i), representing a positive interaction between user u and item i.

The content of the user-item matrix is naturally different for the two above types of feedback. Figure 2.1 illustrates user-item matrices for both ratings and positive-only data. When numeric ratings are available, the main task typically consists of predicting missing values in the user-item matrix. This is a natural formulation when numeric ratings are available, and the problem is naturally seen as a *rating prediction* task. However, some systems employ positive-only ratings. These systems are quite common – e.g. *like*/*favorite* buttons, music

	i_1	i_2	i_3	i_4	...	i_n
u_1	1		5			
u_2			4			2
u_3				3		
u_4		1		5		
...						
u_m			2			

	i_1	i_2	i_3	i_4	...	i_n
u_1	√		√			
u_2			√			√
u_3				√		
u_4		√		√		
...						
u_m			√			

Figure 2.1: Example of feedback matrices. On the left, a typical numerical ratings matrix. On the right a positive-only ratings matrix.

streaming, shopping carts, news reading. In these cases, the matrix R is a boolean value matrix, where $true$ values indicate a positive user preference, and $false$ – typically the vast majority – may indicate one of two things: the user either does not like or does not know the item. In systems with positive-only user feedback, the task is to predict $true$ values in R, which is more closely related to classification problems. This type of feedback is also known in the literature as *binary ratings* or *implicit feedback*. We adopt the term *positive-only*, since the term *binary* may suggest the existence of both positive and negative feedback and the term *implicit* may not be accurate – for instance, clicking a *like* button can hardly be considered an implicit preference. Recommendation using positive-only feedback is also known in the literature as *one-class collaborative filtering* [Pan et al., 2008].

Considering our focus on the *top-N* recommendation task, the type of feedback being used has important implications. For example, when using numeric ratings, a recommendation list can be easily produced by sorting items by descending predicted rating. This, however, is not so trivial when using positive-only data. Generally, we either need to predict some kind of preference level for items – in order to sort them for each user – or to directly approach the task as a learning to rank problem [Liu, 2009].

Most state-of-the-art CF algorithms are based on either neighborhood methods or matrix factorization methods. Fundamentally, these differ on the strategy used to process data in the ratings matrix.

2.3.2 Neighborhood-based algorithms

Neighborhood-based CF algorithms essentially compute user or item neighborhoods using similarity measures such as the cosine or Pearson correlation [Sarwar et al., 2001]. If the rows of R represent users and the columns correspond to items, similarity between two users u and v is obtained using the rows corresponding to those users, R_u and R_v. Similarity between two items i and j can be obtained between the columns corresponding to those items R_i and R_j. Recommendations are computed by searching and aggregating through user or item neighborhoods. The main advantages of neighborhood methods are their simplicity and ease of implementation, as well as the trivial explainability of recommendations – user and item similarities are intuitive concepts. The main downside of neighborhood-based methods is the lack of scalability, since that both time and space complexity grow simultaneously with both the number of users and the number of items in the system.

2.3.2.1 User-based CF

User-based CF exploits similarities between users to form user neighborhoods. Given two users u and v, the similarity between them is given by a measure, typically the Pearson Correlation and the Cosine.

Cosine measure For two users u and v, the cosine measure takes the rows of the ratings matrix R_u and R_v as vectors in a space with dimension equal to the number of items rated by u and v:

$$\mathrm{sim}(u,v) = \cos(R_u, R_v) = \frac{R_u \cdot R_v}{||R_u|| \times ||R_v||} = \frac{\sum_{i \in I_{uv}} R_{ui}\, R_{vi}}{\sqrt{\sum_{i \in I_u} R_{ui}^2}\sqrt{\sum_{i \in I_v} R_{vi}^2}} \tag{2.1}$$

where $R_u \cdot R_v$ represents the dot product between R_u and R_v, I_u and I_v are the sets of items rated by u and v respectively and $I_{uv} = I_u \cap I_v$ is the set of items rated by both users u and v.

A common problem with the cosine is that different users may use the rating scale differently. For example, in a system with a rating scale of integers from 1 to 5, one user can interpret the value 3 as a positive rating, while another user can see it as negative rating. This means that different preference levels can be expressed using the same value. Conversely, equal preference levels may result in different ratings.

To minimize this problem, the Pearson Correlation can be used instead.

Pearson Correlation For users u and v, the Pearson Correlation is given by:

$$\text{sim}(u,v) = \frac{\sum_{i \in I_{uv}} (R_{ui} - \bar{R}_u)(R_{vi} - \bar{R}_v)}{\sqrt{\sum_{i \in I_{uv}} (R_{ui} - \bar{R}_u)^2} \sqrt{\sum_{i \in I_{uv}} (R_{vi} - \bar{R}_v)^2}} \tag{2.2}$$

where $\bar{R}_u$ and $\bar{R}_v$ are the average ratings given by users u and v, respectively. These averages have a normalizing effect on ratings given by different users, minimizing the scale interpretation problem.

Rating prediction To compute a rating prediction $\hat{R}_{ui}$ given by the user u to item i, an aggregating function is used that combines the ratings given to i by the subset $K_u \subseteq U$ of the k users most similar to u – the optimal value of k can be obtained using cross-validation. Two examples of this function are [Adomavicius and Tuzhilin, 2005; Breese et al., 1998]:

$$\hat{R}_{ui} = \frac{\sum_{v \in K_u} \text{sim}(u,v) R_{vi}}{\sum_{v \in K_u} \text{sim}(u,v)} \tag{2.3}$$

$$\hat{R}_{ui} = \bar{R}_u + \frac{\sum_{v \in K_u} \text{sim}(u,v)(R_{vi} - \bar{R}_v)}{\sum_{v \in K_u} \text{sim}(u,v)} \tag{2.4}$$

Equation (2.3) performs a weighted average, in which weights are given by the similarities between u and v. Equation (2.4) incorporates the average ratings given by u and v in order to minimize differences in how users use the rating scale.

2.3.2.2 Item-based CF

Similarity between items can also be explored to provide recommendations [Sarwar et al., 2001; Linden et al., 2003]. The main motivation for the use of item-based algorithms is that many systems have a larger number of users than items. In such systems the dimension of the similarity matrix is significantly reduced using item-based CF. As in user-based algorithms, Cosine and Pearson Correlation can be used as item-based similarity measures:

Cosine measure Let U_i and U_j be the set of users that rated items i and j respectively, and $U_{ij} = U_i \cap U_j$ the set of users that co-rated *both* items i and j. The similarity between i and j is given by the cosine of the angle formed by vectors R_i and R_j, whose coordinates are the ratings given by all users to each of the items:

$$\text{sim}(i,j) = \cos(R_i, R_j) = \frac{R_i \cdot R_j}{||R_i|| \times ||R_j||} = \frac{\sum_{u \in U_{ij}} R_{ui} R_{uj}}{\sqrt{\sum_{u \in U_i} R_{ui}^2} \sqrt{\sum_{u \in U_j} R_{uj}^2}} \tag{2.5}$$

As in the user-based case, the cosine for item-based similarity suffers from the scale interpretation problem – different users have different interpretations of the rating scale. For item-based cosine similarity the adjusted cosine [Sarwar et al., 2001] can be used instead:

$$\text{sim}(u,v) = \frac{\sum_{u \in U_{ij}} (R_{ui} - \bar{R}_u)\,(R_{uj} - \bar{R}_u)}{\sqrt{\sum_{u \in U_i} (R_{ui} - \bar{R}_u)^2} \sqrt{\sum_{u \in U_j} (R_{uj} - \bar{R}_u)^2}} \tag{2.6}$$

In the above equation, the average rating $\bar{R}_u$ given by user u is used to minimize the effect of different interpretations of the rating scale.

Pearson Correlation The Pearson Correlation can also be used in item-based algorithms as a measure of similarity between two items i and j. It is given by:

$$\text{sim}(i,j) = \frac{\sum_{u \in U_{ij}} (R_{ui} - \bar{R}_i)(R_{uj} - \bar{R}_j)}{\sqrt{\sum_{u \in U_{ij}} (R_{ui} - \bar{R}_i)^2} \sqrt{\sum_{u \in U_{ij}} (R_{uj} - \bar{R}_j)^2}} \tag{2.7}$$

where $\bar{R}_i$ and $\bar{R}_j$ are the average ratings given to i and j, respectively.

Rating prediction According to [Sarwar et al., 2001], the prediction of the rating given u to item i is obtained like this: let K_i be the set of items most similar to i. Then, an aggregating function such as one of the two below is used:

$$\hat{R}_{ui} = \frac{\sum_{j \in K_i} \text{sim}(i,j) R_{uj}}{\sum_{j \in K_i} \text{sim}(i,j)} \tag{2.8}$$

$$\hat{R}_{ui} = \bar{R}_i + \frac{\sum_{j \in K_i} \text{sim}(i,j)(R_{uj} - \bar{R}_j)}{\sum_{j \in K_i} \text{sim}(i,j)} \tag{2.9}$$

Equation (2.8) is the weighted average of the ratings given by u to the items similar to i. The similarity values $\text{sim}(i,j)$ are the weighting factors. In (2.9) the average rating given to i and j are used to eliminate eventual biases on how those items are rated. Like in the user-based case, the optimal number of nearest-neighbors K_i is typically obtained via cross-validation.

2.3.2.3 Neighborhood-based CF for positive-only data

The methods in Section 2.3.2 are designed to work with numerical ratings. However, our focus in this thesis is on positive-only data. Neighborhood-based CF for positive-only data can actually be approached as a special case of neighborhood-based CF for ratings, by simply considering $R_{ui} = 1$ for all observed (u, i) user-item pairs and $R_{ui} = 0$ for all other cells in the feedback matrix R. Both notation and implementation can be simplified with this. With positive-only data, the rating scale interpretation problem does not apply. The most practical similarity measure is then the cosine, given by (2.1). In the user-based case, it can be simplified to:

$$\text{sim}(u, v) = \cos(R_u, R_v) = \frac{\sum_{i \in I_{uv}} R_{ui}\, R_{vi}}{\sqrt{\sum_{i \in I_u} R_{ui}^2}\sqrt{\sum_{i \in I_v} R_{vi}^2}} = \frac{|(I_u \cap I_v)|}{\sqrt{|I_u|} \times \sqrt{|I_v|}} \tag{2.10}$$

where I_u and I_v the set of items that are observed with u and j, respectively.

In the item-based case, the simplification is analogous:

$$\text{sim}(i, j) = \cos(R_i, R_j) = \frac{\sum_{u \in U_{ij}} R_{ui}\, R_{uj}}{\sqrt{\sum_{u \in U_i} R_{ui}^2}\sqrt{\sum_{u \in U_j} R_{uj}^2}} = \frac{|(U_i \cap U_j)|}{\sqrt{|U_i|} \times \sqrt{|U_j|}} \tag{2.11}$$

In (2.11), U_i and U_j are the set of users that are observed with i and j, respectively.

The cosine formulation in (2.10) and (2.11) for positive-only ratings allows the calculation of user-user or item-item similarities using simple user occurrence and co-occurrence counts. A user u is said to co-occur with user v for every item i they both occur with. Similarly, an item i is said to co-occur with item j every time they both occur with a user u. A prediction for user u and item i can be made using one of (2.3) or (2.4), in the user-based case, or (2.8), in the item-based case. One important notion is that the value of $\hat{R}_{ui}$ is not a rating, but rather a score between 0 and 1, by which a list of candidate items for recommendation can be sorted in descending order for every user.

2.3.3 Matrix factorization methods

The most studied alternative to neighborhood-based CF is Matrix Factorization (MF). So far, MF methods have proved to be generally superior to neighborhood methods in large scale problems, in terms of both predictive ability and run-time complexity [Shi et al., 2014].

Matrix Factorization for CF was initially inspired by Latent Semantic Indexing [Deerwester et al., 1990], a popular technique to index large collections of text documents, used in the

field of information retrieval. LSI performs the Singular Value Decomposition (SVD) of large document-term matrices. In a CF problem, the same technique can be used in the user-item matrix, uncovering a latent feature space that is common to both users and items. One important feature of SVD is that it provides the ability to make optimal lower rank approximations to the original matrix. One problem with SVD is that classic factorization algorithms, such as Lanczos methods, are not defined for sparse matrices. This issue has been addressed by performing some form of value imputation in the user-item matrix [Billsus and Pazzani, 1998; Sarwar et al., 2000b]. This, however, is a potential source of systematic error, especially taking into account that the user-item matrix in CF problems is typically very sparse.

As an alternative to classic SVD, optimization methods [Bell and Koren, 2007; Funk, 2006; Paterek, 2007; Takács et al., 2009] have been proposed to decompose (very) sparse user-item matrices[1].

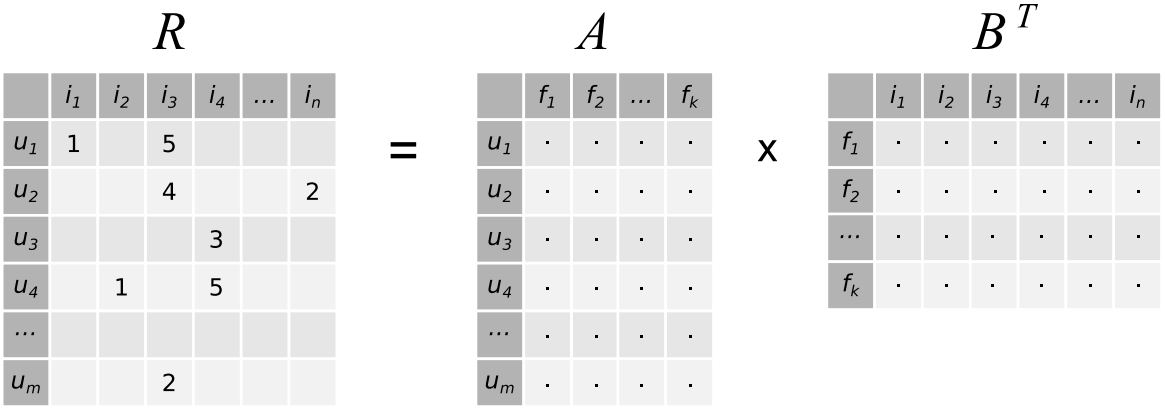


Figure 2.2: Matrix factorization: $R = AB^T$.

Figure 2.2 illustrates the factorization problem. Supposing we have a user-item matrix $R_{m \times n}$ with m users and n items, the algorithm decomposes R in two full factor matrices $A_{m \times k}$ and $B_{n \times k}$ that, similarly to SVD, cover a common k-dimensional latent feature space, such that R is approximated by $\hat{R} = AB^T$. Matrix A spans the user space, while B spans the item space. The k latent features describe users and items in a common space. Given this formulation, the predicted rating by user u to item i is given by a simple dot product:

$$\hat{R}_{ui} = A_u \cdot B_i \tag{2.12}$$

The number of latent features k is a user given parameter that controls the model size. Individual latent features can represent simple and easy to understand concepts, as well

[1] In some of the literature, these methods are often referred to as SVD, despite being formally different methods.

as complex high level concepts and their combinations. For instance, in a movie rating dataset, one latent feature may encode a specific genre – e.g. action, comedy, horror – while another may encode complex features such as Oscar winning movies from the late seventies. The semantics of latent features is not known a priori, however a good algorithm should be able to extract the most important ones for the given data. A good exploratory analysis is done in [Takács et al., 2009], where the authors are able to identify the implicit encoding of genre on latent features.

2.3.3.1 Stochastic gradient descent

The model consists of A and B, so the training task consists of estimating A and B such as their product approximates R as accurately as possible. This is done by minimizing error on the known ratings. Training is performed by minimizing the L_2-regularized squared error for known values in R and the corresponding predicted ratings:

$$\min_{A.,B.} \sum_{(u,i)\in D} (R_{ui} - A_u \cdot B_i)^2 + \lambda_u ||A_u||^2 + \lambda_i ||B_i||^2 \tag{2.13}$$

In the above equation, D is the set of user-item pairs for which ratings are known and λ is a parameter that controls the amount of regularization. The regularization terms $\lambda||A_u||^2$ and $\lambda_i||B_i||^2$ are used to avoid overfitting. These terms penalize parameters with high magnitudes, that typically lead to overly complex models with low generalization power. For the sake of simplicity, we use $\lambda = \lambda_u = \lambda_i$, which results in a single regularization term $\lambda(||A_u||^2 + ||B_i||^2)$. The rank of the factor matrices A and B – the number of latent features k – is a user-defined hyperparameter that can be used to control the trade-off between model size and information loss. The most computationally efficient methods to solve this optimization problem are Alternating Least Squares (ALS) [Bell and Koren, 2007] and Stochastic Gradient Descent (SGD) [Funk, 2006]. It has been shown [Funk, 2006; Paterek, 2007] that SGD based optimization generally performs better than ALS when using very large and sparse datasets – which is typically the case in recommender systems –, both in terms of model accuracy and run time performance. In this thesis, we study the simple case of processing data over a single CPU, however some interesting alternatives to simple SGD are available in the literature that are able to take advantage of parallel processing platforms [Gemulla et al., 2011; Yu et al., 2014] and/or Graphical Processing Units [Rodrigues et al., 2015].

Given a training dataset consisting of tuples in the form (u, i, r) – the rating r of user u to item i –, SGD performs several passes through the dataset, known as iterations or epochs, until some stopping criterion is met, typically a convergence bound and/or a maximum number

of iterations. At each iteration, SGD sweeps over all known ratings R_{ui} and updates the corresponding rows A_u and B_i, correcting them in the opposite direction of the gradient of the error, by a factor of $\eta \leq 1$ – known as step size or learn rate. The algorithm starts by initializing matrices A and B with small random numbers – typically following a gaussian[2] $\mathcal{N}(\mu, \sigma)$ with $\mu = 0$ and and small σ. For each known rating, the corresponding error is calculated as $err_{ui} = R_{ui} - \hat{R}_{ui}$, and the following update operations are performed:

$$\begin{aligned} A_u &\leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u) \\ B_i &\leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i) \end{aligned} \tag{2.14}$$

Algorithm 2.1: BSGD - Batch SGD

Data: a dataset $D = (u, i, r)_1, \ldots, (u, i, r)_n$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

init
  for $u \in$ `Users`(D) **do**
    $A_u \leftarrow$ `Vector`(size : k)
    $A_u \sim \mathcal{N}(0, 0.1)$
  for $i \in$ `Items`(D) **do**
    $B_i \leftarrow$ `Vector`(size : k)
    $B_i \sim \mathcal{N}(0, 0.1)$
for $count \leftarrow 1$ **to** $iter$ **do**
  $D \leftarrow$ `Shuffle`(D)
  for $(u, i, r) \in D$ **do**
    $err_{ui} \leftarrow r - A_u \cdot B_i$
    $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$

Algorithm 2.1 implements this method. It has first been informally proposed in [Funk, 2006] and many extensions have been proposed ever since [Paterek, 2007; Koren, 2008; Takács et al., 2009; Salakhutdinov and Mnih, 2007]. One obvious advantage of SGD is that complexity grows linearly with the number of known ratings in the training set, actually taking advantage of the high sparsity of R.

[2] We use $\mathcal{N}(0, 0.1)$ in all algorithms.

Other proposed factorization methods include probabilistic Latent Semantic Analysis (pLSA), used in [Hofmann, 2004] and [Takacs et al., 2007], and CF via Principal Component Analysis (PCA) [Goldberg et al., 2001].

2.3.4 Matrix factorization for positive-only data

Algorithm 2.1 (BSGD) is designed to work with ratings data. The input of the algorithm is a set of triples in the form (u, i, r), each corresponding to a rating r given by a user u to an item i. It is possible to use BSGD with positive-only data by simply assuming that $r = 1$ for all cases. This results in Algorithm 2.2.

Algorithm 2.2: BSGD - Batch SGD for positive-only data

Data: a dataset $D = (u, i)_1, \ldots, (u, i)_n$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

```
1  init
2      for u ∈ Users(D) do
3          A_u ← Vector(size : k)
4          A_u ~ N(0, 0.1)
5      for i ∈ Items(D) do
6          B_i ← Vector(size : k)
7          B_i ~ N(0, 0.1)
8  for count ← 1 to iter do
9      D ← Shuffle(D)
10     for (u, i) ∈ D do
11         err_ui ← 1 − A_u · B_i
12         A_u ← A_u + η(err_ui B_i − λ A_u)
13         B_i ← B_i + η(err_ui A_u − λ B_i)
```

The only differences between Algorithms 2.1 and 2.2 are the type of input data and the error calculation. Algorithm 2.2 receives pairs in the form (u, i) and assumes $r = 1$, causing err_{ui} to be calculated as the difference to 1 always. In the end, the predicted "ratings" $\hat{R}_{ui} = A_u.B_i$ will be a value indicating a user's preference level for an item. This value can be used in a sorting function f to order a list of items:

$$f_{ui} = |1 - \hat{R}_{ui}| \tag{2.15}$$

Equation (2.15) measures the proximity of a predicted rating to 1. If we look at BSGD, it is evident that what it models is exactly this. A_u and B_i are always adjusted to minimize the error with respect to 1, so it is natural to assume that the most relevant items for a user u are the ones that minimize the difference above. Note that since we are not imposing on the model any restrictions to the prediction values, which means that $\hat{R}_{ui}$ is not restricted to the interval $[0, 1]$. This is why we use the absolute value of the difference in (2.15).

One important limitation of using BSGD with positive-only data is that it could result in trivial models. If we use a model where $\hat{R}_{ui} = 1$ for all (u, i) pairs, the error would always be $err_{ui} = 1 - \hat{R}_{ui} = 0$, and no learning would be performed. This does not happen in practice for two reasons. First, regularization squashes down the values of feature vectors with high magnitude, which in practice forces predictions to always be $\hat{R}_{ui} < 1$. Second, we initialize the feature vectors with random values close to 0, which forces the model to always begin a learning process. Although regularization and model initialization may help prevent trivial solutions, they do not entirely solve the problem. When learning from large datasets – which is a common scenario in recommender systems – most predictions will tend to accumulate closely together, eventually leading to a model with low discriminative power. One solution is to try to counterbalance the model with negative examples that can be inferred from the positive ones. We discuss this technique in Chapter 4. Another solution is to approach the recommendation problem directly as a learning-to-rank problem.

2.3.5 Learning to rank - BPRMF

If we look at the *top-N* recommendation problem, we are in fact trying to obtain a list of ranked items. Learning-to-rank algorithms [Liu, 2011] directly model the relative position between items in a list. In [Rendle et al., 2009], Rendle et al. use a Bayesian framework to capture personalized item rankings. Given the set of users U and the set of items I in a dataset consisting of positive user-item interactions (u, i), the task consists of finding an optimal item ranking $>_u$ for each user – i.e. a personalized ranked set of items. This set follows the three poperties of a total order. For any user u, the personalized rank I_u of any items i, j and k in I is such that:

- if $i >_u j$ and $j >_u i$ then $i = j$ (antisymmetry)
- if $i >_u j$ and $j >_u k$ then $i >_u k$ (transitivity)
- $i >_u j$ or $j >_u i$ for all i, j (totality)

Instead of scoring items for users, Rendle et al. devise a ranking model, that establishes a personalised rank of items for each user $>_u$, by assuming that all items i observed for that user – the observed pairs (u, i) in the dataset – precede, in $>_u$, all items j that do not occur with u in the dataset. The Bayesian formulation – Bayesian Personalize Ranking (BPR) – consists of maximizing the posterior probability $p(\Theta|\ >_u)$, where Θ represents the parameters of an arbitrary model. This formulation is especially helpful because it is independent of the predictive model. Effectively, BPR is applicable to both neighborhood and factorization models. We will focus on BPR with Matrix Factorization – BPRMF. In this case, $\Theta = (A, B)$, with A and B being the two factor matrices representing user and item latent features respectively. Learning is performed using triples in the form (u, i, j), that correspond to the actual observed pair (u, i) with an additional item j sampled from the set of items not observed with u. Predictions $\hat{R}_{uij}$ indicate the relative rank of i and j, which is expressed by the difference between the dot products:

$$\hat{R}_{uij} = A_u \cdot B_i - A_u \cdot B_j \tag{2.16}$$

This allows the use of conventional matrix factorization, together with SGD to train a ranking model. The update operation for the parameter set Θ in the SGD algorithm is:

$$\Theta \leftarrow \Theta + \eta(\sigma(\hat{R}_{uij})\frac{\partial}{\partial\Theta}\hat{R}_{uij} + \lambda_\Theta\Theta) \tag{2.17}$$

where η is the learn rate, σ is the logistic function $\sigma(x) = \frac{e^{-x}}{1+e^{-x}}$ and λ_Θ is a set of regularization factors for the parameters. In the matrix factorization algorithm, three parameter vectors are updated at each iteration:

$$\begin{aligned} A_u &\leftarrow A_u + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)(B_i - B_j) + \lambda_u A_u) \\ B_i &\leftarrow B_i + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_i B_i) \\ B_j &\leftarrow B_j + \eta(-\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_j B_j) \end{aligned} \tag{2.18}$$

Note that three different regularization factors λ_u, λ_i and λ_j need to be set. Algorithm 2.3 is the complete implementation of BPRMF.

In Algorithm 2.3, k, $iter$, η and the three $\lambda_{\{u,i,j\}}$ are respectively the number of latent features, the number of iterations, the learn rate and the regularization factors for users, observed items and unobserved items. The `SampleFrom` typically performs uniform sampling from the set of items that the user has not interacted with. A variant of this algorithm, that is known as Weighted BPRMF – or WBPRMF – performs non-uniform sampling from the same set, by sampling items with probability proportional to their popularity.

Algorithm 2.3: BPRMF - Bayesian Personalized Ranking Matrix Factorization

Data: a dataset $D = (u, i)_1, \ldots, (u, i)_n$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ_u, λ_i and λ_j the regularization factors
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

init
  for $u \in \texttt{Users}(D)$ **do**
    $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
    $A_u \sim \mathcal{N}(0, 0.1)$
  for $i \in \texttt{Items}(D)$ **do**
    $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
    $B_i \sim \mathcal{N}(0, 0.1)$
for $count \leftarrow 1$ **to** $iter$ **do**
  $D \leftarrow \texttt{Shuffle}(D)$
  for $(u, i) \in D$ **do**
    $j \leftarrow \texttt{SampleFrom}(\{j|(u, j) \notin D\})$
    $A_u \leftarrow A_u + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)(B_i - B_j) + \lambda_u A_u)$
    $B_i \leftarrow B_i + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_i B_i)$
    $B_j \leftarrow B_j + \eta(-\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_j B_j)$

2.3.6 Other methods

Although the majority of research on algorithms for recommender systems is focused on either neighborhood or matrix factorization methods, other alternatives have been proposed to solve CF problems. Probabilistic strategies, such as Clustering and Bayesian networks, have been presented in early work [Breese et al., 1998]. Clustering is motivated by the assumption that it is possible to group users in clusters according to their preferences. In [George and Merugu, 2005; Symeonidis et al., 2006; de Castro et al., 2007], *co-clustering* – or *bi-clustering* – simultaneously clusters users and items, with gains in computational performance and without significant accuracy loss. The reasoning behind co-clustering it that it is frequent that users in a cluster prefer specific subsets of items.

Other probabilistic approaches are based on Graph Theory [Aggarwal et al., 1999; Huang et al., 2002, 2004; Fouss et al., 2007; Gori and Pucci, 2007; Tiroshi et al., 2014]. For systems with binary ratings, Association Rules [Sarwar et al., 2000a; Mobasher et al., 2001; Lin et al., 2002] and Markov Chains [Shani et al., 2005; Rendle et al., 2010] have also been proposed.

In [Khoshneshin and Street, 2010], the MF problem is reformulated as an Euclidean embedding problem that joins users and items in a common Euclidean space. The model is learned with SGD in a analogous process to MF and Euclidean distances between users and items are directly used to predict ratings.

2.4 Hybrid recommenders

There is no fundamental constraint on combining content-based filtering with collaborative filtering. A third type of recommender systems, known as hybrid recommendation, does exactly that, taking advantage of both collaborative filtering and content-based filtering. Fundamentally, hybrid recommender systems vary in the strategy used to combine recommendation methods from the two worlds, for which there is a considerable variety of approaches [Burke, 2002, 2007]. Usually, the main motivation of using hybrid methods is to compensate the limitations of collaborative filtering, with content-based recommendation techniques. One important limitation of CF algorithms is known as the cold-start problem [Adomavicius and Tuzhilin, 2005], which occurs when there is not enough information to compute a reliable recommendation model. This can happen globally, e.g. when kickstarting a system, or individually for new users or items that enter the system and still do not have enough activity. Content-based recommendation can be used in these early stages, to compensate for the lack of predictive ability of the CF model.

2.5 Context-aware collaborative filtering

In many applications, *context* can be very relevant. For example, the music a user listens to when exercising may be completely different from the music she listens to while reading. Another good example is Point-Of-Interest (POI) recommendation, where the current user location is crucial to generate useful recommendations. Moreover, many times, the available data is not limited to user-item interactions – be them ratings or positive-only feedback. Context-aware recommender systems try to exploit context features to improve recommendations.

In the context-aware framework, recommender systems can use context features in three ways, identified by Adomavicius et al. in [Adomavicius et al., 2011]:

1. *Pre-filtering*: filter input data considered by the recommender according to the context in which recommendation is requested. One extreme example is to ignore all the user's feedback given in weekdays when she requests recommendations for weekends. In general, pre-filtering is achieved by modifying the input data before training the model;

2. *Post-filtering*: filter out irrelevant items from recommendations originally produced by a conventional context-unaware model. This generally works by filtering out from conventional recommendations the items that do not match the context for which the recommendation is requested;

3. *Modeling*: explicitly model the context features at the learning stage. Here the model should be able to produce adequate recommendations according to the context for which recommendations are generated.

A considerable body of work is available specifically on context-aware recommendation, although most of it is beyond the scope of this overview. In the following section, we will only refer to those works that deal specifically with time.

2.6 Time-aware and time dependent collaborative filtering

The relevance of time information in recommender systems is based on the assumption that one or more concepts modeled by recommender systems – e.g. user preferences, item popularity – naturally change over time. Traditional CF algorithms do not account for this and need to be frequently retrained to adjust to time related phenomena. For example, travel destination recommendation models would probably need to be retrained to adjust

to the current season. This has motivated researchers to investigate new techniques that automatically adjust models to time-evolving phenomena, therefore avoiding complicated maintenance of recommendation models. The main distinction between algorithms that deal with time lies on how the time dimension is approached.

In [Vinagre et al., 2015b], we provide an overview of the main contributions in temporal CF. In another recent survey on temporal CF [Campos et al., 2014], Campos et al. review recommendation algorithms that deal with time, emphasizing evaluation. In this section, we focus on the most relevant time-related algorithms.

2.6.1 Approaching the time dimension

One strategy to exploit time is to use it as context information when training recommendation models. Using time as context, CF algorithms use timestamps as an additional source of information, thereby enriching the model. The natural reasoning behind this strategy is that users tend to repeat habits at regular time intervals or moments. By capturing the time at which user preferences are observed in the past, time-aware algorithms make better predictions in similar time patterns occurring in the future. For example, when requesting movie recommendations for the weekend, predictions would be expected to match the general preferences of the user during weekends. These predictions would possibly be different from those that would have been made for weekdays. Following the terminology used in [Shi et al., 2014], we refer to algorithms using this approach as *time-aware* algorithms. Generally, they are a special case of *context-aware* algorithms [Adomavicius et al., 2011], in which the context is given by some kind of temporal information, such as the time of day, day of week or other similar time feature. This information is typically exploited to adjust recommendations to the time for which they are requested.

Another way to approach time is to look at user preference data as a chronologically ordered sequence, such as a time series or a data stream. For this approach, timestamps are not strictly required, since time information itself is not necessarily used. Instead, the approach is to train the model in a way that the chronological order is captured and used to improve predictions. Model training is preferably – but not necessarily – performed in incremental steps, and some kind of recency-based modeling scheme can be used to tackle time-varying concepts. As in [Shi et al., 2014], we refer to algorithms using this approach as *time-dependent* algorithms, although terms such as *sequence-aware* CF or simply *sequential* CF can be used, since these are algorithms that exploit user feedback sequentially. Within this approach we also identify some contributions in which time is used to categorize the users' preferences in terms of their temporal stability. If one considers that users have some preferences more persistent than others, these can be explicitly modeled as long-term and short-term preferences.

2.6.2 Time-aware algorithms: time as context

Time-aware algorithms specifically use time-related information as context. Typical time features are time of day, day of week, working/non-working hours and seasonal information – e.g. winter/summertime, holidays. Generally time-aware recommendation is a special case of context-aware recommendation, using time-related context features.

2.6.2.1 Time-aware factorization models

Tensor decomposition models Tensors are the n-dimensional generalization of matrices. By adding an extra dimension to the ratings matrix and by projecting time on that new dimension, a 3-dimensional tensor can be obtained. By factorizing this tensor, it is possible to model ratings according to the time at which past ratings occurred – a time modeling approach. The assumption is that users tend to have cyclic habits, – e.g., watching comedies on sundays, or listening to classical music in the evening. Let $R \in \mathbb{R}^{|U|\times|I|\times|T|}$ be a three-dimensional tensor where dimensions span U, I and T, respectively the set of users, set of items and set of time features – e.g time of day, day of week. There is a considerable number of methods to perform the actual decomposition [Kolda and Bader, 2009], however the CANDECOMP/PARAFAC (CP) decomposition model has shown to be well suited for sparse tensors [Acar et al., 2010]. The CP decomposition model is illustrated in Fig. 2.3. Three factor matrices can be obtained, each spanning one of the tensor's dimensions on a set of k latent features, by minimizing the prediction error on known ratings. The tensor can then be approximated as ($\circ$ denotes the outer product):

$$\hat{R} = \sum_{d=1}^{k} U_d \circ I_d \circ T_d \tag{2.19}$$

In the 2010 Challenge in Context-aware Movie Recommendation – CAMRa2010 [Said et al., 2010] –, one of the tasks consisted of recommending movies for specific weeks. Two contributions to this challenge rely on tensor factorization. In [Gantner et al., 2010], Gantner et al. use a tensor factorization model (Pairwise Interaction Tensor Factorization - PITF) originally developed for tag recommendation [Rendle and Schmidt-Thieme, 2010]. The PITF model separately performs pairwise factorizations between every two dimensions. Formally,

$$\hat{R} = \sum_{d=1}^{k} U_d \cdot I_d + \sum_{d=1}^{k} U_d \cdot T_d + \sum_{d=1}^{k} I_d \cdot T_d \tag{2.20}$$

The authors use weekly time-bins – with some overlapping – according to the week of rating.

By adding this new dimension to the user-item matrix and obtaining the corresponding tensor factorization the model is capable of taking advantage of the temporal context. Despite this, the method was unable to outperform a state-of-the-art time-agnostic algorithm [Rendle et al., 2009] that does not use time information.

Also within CAMRa2010, in another contribution [Liu et al., 2010a], Liu et al. propose two time-aware methods for the same task in the competition. The first method is based on CP tensor factorization – by adding time bins as a dimension to the user-item matrix – and the other is a sequential matrix factorization model that consists of several factorizations, one for each time bin – a pre-filtering method. Both methods were successful in outperforming baselines that consisted of time-agnostic versions of the proposed algorithms. In [Liu et al., 2013], Liu et al. provide a more detailed description of sequential matrix factorization and show that combining temporal and social network context information, enables the method to outperform state-of-the-art time-independent algorithms.

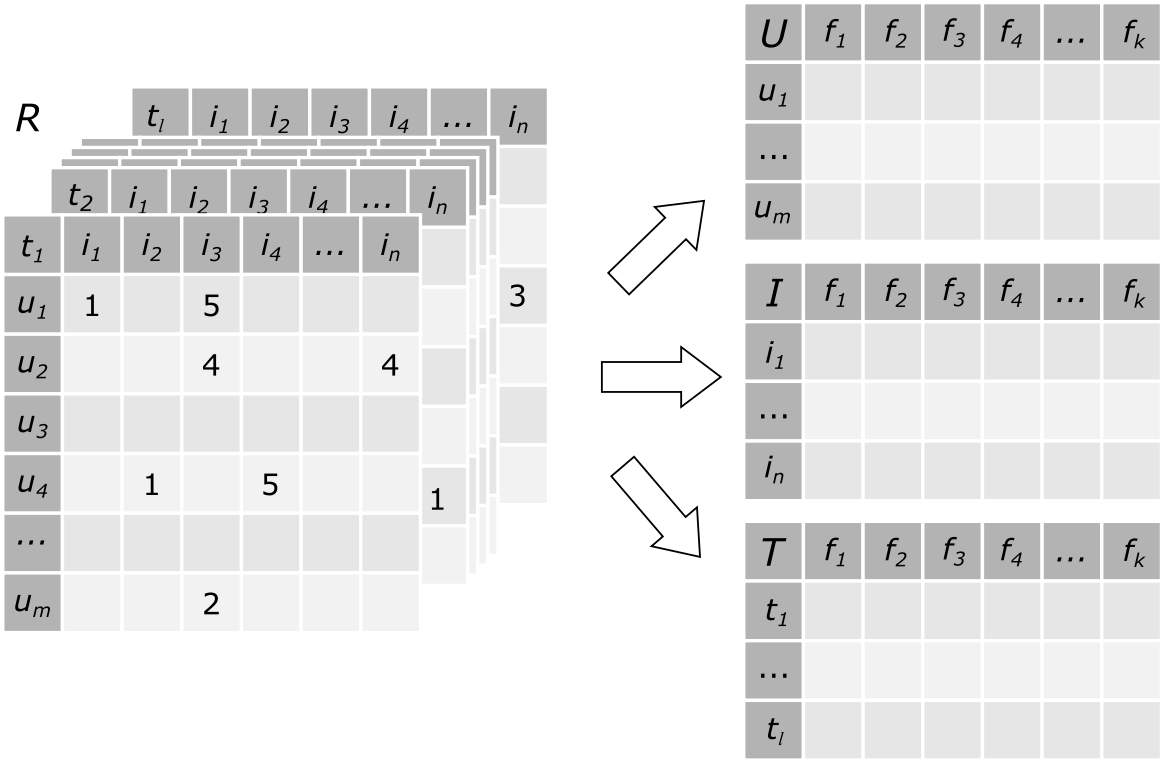


Figure 2.3: CANDECOMP/PARAFAC tensor factorization model: three matrices are obtained covering the same latent feature space $f_{1...k}$. The time-related dimension $t_1 \dots t_l$ is a time feature extracted from the ratings timestamp – e.g. day of week, month, hour of day. Every cell in the original tensor can be predicted using the inner product of the three corresponding vectors $\hat{R}_{uit} = \sum_{d=1}^{k} U_{ud} I_{id} T_{td}$.

Other time-aware factorization models In [Baltrunas and Amatriain, 2009], Baltrunas and Amatriain use a time-aware factorization algorithm that subdivides user profiles in *micro-profiles*. Each micro-profile is a representation of a user within a specific time frame – e.g. weekend, weekday, morning, winter. One advantage is that pre-filtering can be performed using overlapping criteria. For example, recommendations for a user in a summer weekend morning can be based on three – *summer, weekend, morning* – micro-profiles. One surprising finding in this work is that the best improvement is achieved when using an apparently meaningless micro-profile – even vs odd hours –, which actually challenges the authors' reasoning.

A different approach is followed by Gao et al. in [Gao et al., 2013]. The authors use non-negative matrix factorization using the time of day on a Point-Of-Interest (POI) recommender system. POI are typically geo-referenced locations that are of potential interest to users, such as monuments in a town, sightseeing spots or parking areas. The algorithm is able to build a model that is sensitive to check-in times in referenced locations. A total of 24 user factor matrices are obtained by factorizing separately for each of the day's 24 hours, while using the same location factor matrix throughout the learning process. Additionally, consecutiveness is exploited by means of a regularization term that penalizes predictions based on distant times of day. Recommendations are obtained by aggregating recommendations obtained from each of the 24 time-based sub-models. The algorithm is able to considerably outperform the baseline time-agnostic factorization method as well as a classic neighborhood-based algorithm.

2.6.2.2 Time-aware neighborhood models

Youan et al. propose a time-aware Point-Of-Interest (POI) recommendation system in [Yuan et al., 2013], by incorporating the time-of-day information in the cosine similarity calculation between users in a classic user-based CF algorithm. This explicit modeling approach is based on the assumption that users that check-in in the same locations *at the same time of day* share a higher similarity than users that check-in in the same places but at different times of the day. Additionally, the authors use a probabilistic model to capture popularity patterns over time – e.g. restaurants at meal time, theaters in the evening. Results suggest that modeling the time context for POI recommendation can significantly improve accuracy.

2.6.2.3 Other time-aware contributions

There are several other examples of algorithms that use time as context available in the literature on context-aware recommendation. Some examples are [Adomavicius et al., 2005; Domingues et al., 2008; Panniello et al., 2009; Gorgoglione and Panniello, 2009;

Lee et al., 2010; Panniello et al., 2014]. We do not elaborate on these since they use broader contexts in which time is not studied separately from the other context features, and therefore the impact of the usage of time features is not evaluated.

2.6.3 Time-dependent algorithms: time as sequence

Time-dependent algorithms try to capture the phenomena related to continuous temporal dynamics. These phenomena encompass individual user preference changes, drifts in item popularity, fluctuations in activity rates and virtually any temporal effect that may underlie in sequential usage data. Unlike time-aware algorithms, the main objective is not to model cyclic phenomena, but rather to be able to adjust to unprecedented changes. A large variety of contributions have been made to deal with this. Like time-aware algorithms, both neighborhood and factorization time-dependent algorithms have been proposed and/or adapted. Additionally, given the approach to data as a chronologically ordered sequence, there is a natural approximation to the field of data stream mining. As a result, algorithms designed for streaming data have been used or adapted for recommendation tasks as well. Combined short-term and long-term user modeling have also been used to tackle natural variations in users' preferences. We also reference two data pre-processing approaches that drive time-agnostic algorithms to reflect time-dependency, and another approach that embeds users, items and time in a common Euclidean space.

2.6.3.1 Time-dependent neighborhood models

One simple way to adapt neighborhood-based algorithms to temporal effects is to give more relevance to recent observations, and less to past observations, based on the assumption that recent data is more representative of the current reality. This can be achieved using a series of techniques, most of which are based on either discrete time windows [Nasraoui et al., 2007; Lathia et al., 2009; Vinagre and Jorge, 2012] or continuous decay functions [Ding and Li, 2005; Liu et al., 2010b; Vinagre and Jorge, 2012].

Decay function algorithms The main idea of using a decay function is to reduce the importance of past data *gradually*. An example of a time-decay function is:

$$f(t) = e^{-\alpha t} \qquad 0 < \alpha \leq 1 \tag{2.21}$$

In (2.21) t is the time elapsed since a given moment in time. The longer the time, the lower the function value. The parameter α controls the amount of decay.

Early work on time-dependent neighborhood-based CF is presented by Ding and Li in [Ding and Li, 2005], in an item-based algorithm. The authors use (2.21) in the recommendation step, giving higher weight to recently rated items. The rating prediction for an item is weighted by the most recent ratings given to similar items. Formally, rating prediction R_{ui} is obtained by using $f(t)$ to weight the similarities between the candidate item and its neighbors $\mathrm{sim}(i,j)$ (J is the set of top-k neighbors). In practice, recently rated items have a stronger weight than those rated longer ago.

$$\hat{R}_{ui} = \frac{\sum_{j \in J}(\mathrm{sim}(i,j) R_{uj} f(t))}{\sum_{j \in J}(\mathrm{sim}(i,j) f(t))} \tag{2.22}$$

In later work, the same authors propose in [Ding et al., 2006] a recency-based weighting method. The rating prediction is weighted by the most recent ratings given to similar items.

$$\hat{R}_{ui} = \frac{\sum_{j \in J}(\mathrm{sim}(i,j) R_{uj} W_j)}{\sum_{j \in J}(\mathrm{sim}(i,j) W_j)} \tag{2.23}$$

where the weight $W_j = 1 - \frac{|R_{uj} - R_{cj}|}{M}$, with r_{cj} being the most recent rating given to item j and M the maximum value in the rating scale.

In [Liu et al., 2010b], Liu et al. introduce decaying time functions in both the similarity computation and the rating prediction steps of an item-based algorithm. The authors use (2.21) in the similarity computation which, in practice, causes pairs of items to be less and less similar as their ratings are given farther apart in time. At the rating prediction step a similar decay function $g(t) = e^{-\beta t}$ is used to make rating predictions, using the same method as in [Ding and Li, 2005]. The only difference between f and g are the decay parameters α and β – which can optionally be the same.

Given two items i and j:

$$\mathrm{sim}(i,j) = \frac{\sum_{u \in U_{ij}} R_{ui} f(t_{ui}) \; . R_{uj} f(t_{uj})}{\sqrt{\sum_{u \in U_i} (f(t_{ui}) R_{ui})^2} \cdot \sqrt{\sum_{u \in U_j} (f(t_{uj}) R_{uj})^2}} \tag{2.24}$$

In the above equation U_i and U_j are the sets of the users that rated items i and j respectively and $U_{ij} = U_i \cap U_j$. The times t_{ui} and t_{uj} are the times at which user u rated items i and j, respectively. This penalizes similarities between pairs of items when large time intervals occur between the corresponding ratings.

In [Vinagre and Jorge, 2012] Vinagre and Jorge apply a decay function in incremental user-based and item-based neighborhood algorithms for binary ratings. This is done by multiplying the frequencies of items by a constant factor $\alpha < 1$ at each incremental step, when new data is processed. In practice the result is the same as applying a decay function at the similarity computation step in non-incremental algorithms, but retaining the scalability benefits of incremental algorithms.

Sliding-window algorithms Sliding-window algorithms work by considering only data in a window (sliding window) that contains either the latest N instances – for example, the latest 1000 ratings – or the instances contained in the latest time interval – for example, all ratings given in the last 24 hours.

A user-based neighborhood algorithm is proposed by Nasraoui et al. in [Nasraoui et al., 2007]. The algorithm uses a sliding window containing a fixed number of instances. The algorithm computes similarities between the latest user sessions. Each user session consists of a number of ratings given by a user in a short period of time – for instance, during 1 hour.

A different approach is made by Lathia et al. [Lathia et al., 2009] using time intervals. The authors use a set of item-based algorithms differing only in the number of nearest neighbors considered to predict ratings. Algorithms are retrained at fixed 7 day intervals with data in the last interval. Error is continuously monitored for all algorithms, and the algorithm with the lowest error so far is selected to provide recommendations.

Vinagre and Jorge [Vinagre and Jorge, 2012] present a user-based and an item-based algorithm that compute similarities using the latest n user sessions. These algorithms use a bounded and approximately fixed amount of data to rebuild the similarity matrix, with obvious scalability improvements, but still far worse than the run-time performance of incremental algorithms.

Other approaches Min and Han [Min and Han, 2005] use a mixture of item hierarchies, user clustering and time-weighted correlation to improve product recommendations.

2.6.3.2 Time-dependent factorization models

Matrix Factorization (MF) algorithms have also been adapted to deal with sequential data. In [Koren, 2009], Koren extends his SVD++ algorithm [Koren, 2008] to tackle temporal dynamics. This is done by considering the model's time dependent variables as time functions. In the original SVD++ algorithm, Koren separates the model in a baseline model that relies solely in user and item biases b_u and b_i – individual deviations from the global average rating μ – and the factor model, that captures the actual user-item preferences, based on (2.13), with the addition of implicit data extracted from the same dataset. In the time-dependent algorithm both the user/item biases and the factor model are approached as time functions. To model temporal item biases, time is split in discrete time windows (bins) and for user biases, a decay function is used. Another decay function is used to weigh past ratings at the prediction step. According to the author, time-varying item biases capture item popularity changes, time-varying user biases capture the variations in how individual

users use the rating scale and time-varying factors essentially capture the actual preference changes. Empirically, this time-dependent algorithm significantly improves accuracy with the well-known Netflix dataset.

Tensor factorization revisited Tensor factorization has been also proposed for time-dependent recommendations. Here, the tensor's time dimension contains actual time intervals, so the tensor can be viewed as a periodic collection of ratings over time. The time dimension essentially captures the latent features' trends over time.

In [Xiong et al., 2010], Xiong et al. use a CP tensor factorization model by adding a time dimension to the user-item matrix, splitting time in equal length intervals. The authors use Probabilistic Matrix Factorization algorithms studied in [Salakhutdinov and Mnih, 2007] and [Salakhutdinov and Mnih, 2008] to estimate optimal hyper-parameters. Another contribution using CP tensor factorization is made by Rafailidis and Nanopoulos in [Rafailidis and Nanopoulos, 2014]. The authors use a smoothing factor based on the observed levels of preference change to weigh down the corresponding user-item interactions in previous time intervals. In practice, this scheme produces a tensor where past preferences are given less importance in the same measure as the user changes her habits.

Other temporal factorization models Another approach was presented by Das et al. [Das et al., 2007]. The proposed algorithm combines a neighborhood model with pLSA and MinHash clustering [Indyk, 1999] using the parallel computation with MapReduce [Dean and Ghemawat, 2004] in a news recommender. In their work, the authors introduce a time-decaying function in click-rates made by each cluster in news items and a time-based window on co-visitation of news.

In [Matuszyk and Spiliopoulou, 2014] Matuszyk and Spiliopoulou propose and evaluate several selective forgetting strategies for incremental matrix factorization algorithms with ratings data. In [Matuszyk et al., 2015], Matuszyk et al. extend the study with several other forgetting methods and test them with positive-only data in addition to ratings data. In both publications, the authors show that selectively forgetting *some* of the past users' feedback is beneficial to the system. Also using incremental matrix factorization, Vinagre et al. [Vinagre et al., 2015a] use a recency-based scheme to artificially introduce negative feedback data, tackling a known problem [Pan et al., 2008; Hu et al., 2008] that arises from the absence of negative feedback.

Pálovics et al. [Pálovics et al., 2014] use the social influence between users in a social network to improve recommendations. Social influence is modeled using common preferences shown close together in time by two users connected in the social graph.

2.6.3.3 Short-/Long-term preference modeling

Another way to approach sequentially ordered data is to model the users' short-term and long-term preferences separately. Conceptually, this means that each user has potentially two models, one for long-term preferences and another for short-term preferences. In [Ricci et al., 2003], Ricci et al. identify the importance to deal with the duality of short-term preferences – goal oriented and highly dependent on context – and long-term preferences – durable and stable. A hybrid – both content-based and CF-based – recommender is developed and evaluated online with real-users, showing a considerable reduction on the users' effort in finding travel-related products.

Xiang et al. [Xiang et al., 2010] propose a graph-based model to capture the users' short-term and long-term preferences. User nodes in the graph – connected to all the user's preferred items – encode long-term preferences, and *session* nodes, which are time-restricted, encode the user's short-term preferences. Items that match both the long-term and the short-term preferences of a user are recommended. In [Hong et al., 2012], Hong et al. explicitly model short-term user profiles by using consecutive stages, which correspond to periods of time where the user's shopping habits are dominated by items belonging to set of categories within an existing taxonomy. Long-term user preferences are given by multiple stages. The authors use clustering-based and graph-based recommendation algorithms to exploit stage information, outperforming other time-dependent algorithms.

Jannach et al. [Jannach et al., 2013] also emphasize the importance of short-term preferences by re-ordering recommendation lists using short-term preference data, with considerable accuracy gains in an e-commerce dataset.

By maintaining a model consisting of an offline component and an online component, Liu and Aberer [Liu and Aberer, 2014] are able to capture long-term influences – offline component – and short-term preferences – online component. The online component is updated frequently with fresh incoming data, and is therefore more sensitive to context and short-term infulences, while the offline component, containing more stable preferences, is updated much less frequently using the data meanwhile stored in the online compontent. Using this approach, combined with contextual text reviews, the proposed method is able to outperform other state-of-the-art time-dependent algorithms on a dataset extracted from a large ratings website.

2.6.3.4 Data-stream algorithms

In [Li et al., 2007], Li et al. propose an approach to drifting preferences of individual users using the CVFDT algorithm [Hulten et al., 2001]. This is a popular classification algorithm for

high speed data streams that automatically adapts to concept drifts. The CVFDT algorithm is used to build a decision tree for each item in the dataset, given the ratings of other highly correlated items. The ratings given by users to these correlated items are used to predict the ratings for the target item. The algorithm can be extended to use item hierarchies – if they exist – with considerable improvements. The mechanics of CVFDT provides automatic adjustment to drifts in user interests, avoiding accuracy degradation.

In the previously mentioned work by Nasraoui et al. [Nasraoui et al., 2007] a second algorithm uses the TECHNO-STREAMS stream clustering algorithm [Nasraoui et al., 2003], using a sliding window through user sessions.

2.6.3.5 Data pre-processing

Two time-dependent methods in the literature consist of data pre-processing techniques – rather than algorithms. The objective is to encode temporal or sequence information in the data itself, which can therefore be used with any time-agnostic algorithm. In [Zimdars et al., 2001], Zimdars et al. approach the recommendation as a univariate time series problem, an apply two data transformations that encode sequence in the data. Using a decision tree model, the authors are able to improve accuracy over the baseline that ignores data order. Cao et al. use a data pre-processing approach in [Cao et al., 2009], which the authors claim to be possible to use with any algorithm. The process consists of identifying four common user behavior patterns and manipulating user data according to the detected pattern. The authors identify one of the patterns as being noisy behavior and remove data generated according to this pattern. Additionally, some pruning is performed on data generated by users identified as having drifting preferences, by retaining the latest interest. The technique is evaluated using neighborhood-based algorithms.

2.6.3.6 Euclidean embedding

A different proposal is made by Yin et al. in [Yin et al., 2012], as an extension to the Euclidean embedding framework proposed in [Khoshneshin and Street, 2010], where users and items are embedded in the same Euclidean space. By adding time factors to that user-item Euclidean space, the authors claim superior accuracy over the baseline algorithm.

2.6.4 Algorithms both time-aware and time-dependent

Theoretically, time-aware and time-dependent are not mutually exclusive approaches. However we only found a single contribution that encompasses both techniques. This is done

by Campos et al. [Campos et al., 2010] in the context of the aforementioned CAMRa2010 [Said et al., 2010] competition, where the authors use a user neighborhood algorithm that computes recommendations considering recently rated items in the neighborhoods – a time-dependent technique – and ratings given on the same months and days in previous years – in a pre-filtering time-aware technique.

2.6.5 Discussion

All time-aware and time-dependent contributions described in this section significantly improve the predictive ability of algorithms. This is clear in the various comparisons between algorithms capable of capturing temporal dynamics and the equivalent algorithms without this capability. This means that adding time-awareness or time-dependency to algorithms that do not have it is clearly beneficial. However, it is also shown by some authors that some state-of-art algorithms without temporal capabilities are quite hard to outperform. Until these algorithms are given the ability to deal with the dynamics of time – and compared with the respective baselines –, it will be hard to adopt them as state-of-the-art algorithms.

Unfortunately, it is impossible to make a reliable and thorough comparative assessment between all the algorithms. In most cases, this is because the problems being tackled are fundamentally different, leading to completely different experiment designs and evaluation methodologies. Results may also differ considerably by small implementation differences [Said and Bellogín, 2014]. The choice of evaluation protocols and metrics are still subject to an active debate in the recommender systems research community, and naturally researchers will likely choose the methodology that adapts best to their own particular circumstances. The limited amount of available datasets for recommender systems, especially with the additional constraint of having to be either timestamped or chronologically ordered – in order to exploit time – is a major limitation for effectively benchmarking algorithms. In our work, particularly in this thesis, we try to use as many datasets as possible. This has allowed us to detect problems as well as to not generalize about the superiority of some algorithms over others, since this many times is highly dependent on the data being used.

One striking aspect of most of the aforementioned work – and perhaps a relevant research issue – is that run time performance and scalability are somewhat overlooked in the majority of the presented work. While the accuracy of CF algorithms are undoubtedly fundamental, scalability and run time complexity are also major issues in this field of research, and can be decisive factors in the choice of a recommender system, in practice. One more general remark about accuracy, and one that has been debated in the community is the scope of accuracy results obtained in offline experiments, and how it translates – or not – into overall quality from the users' perspective [McNee et al., 2006; Pu et al., 2012]. This is a complicated issue, since a thorough realistic evaluation typically requires access to large

scale production systems, which is not widely available to researchers, for obvious reasons.

Because temporal CF is still a recent topic, most real-world systems still rely on static CF. This means that models have to be frequently retrained which can be a complex task given the large amounts of data. Temporal CF has the potential to improve accuracy by better reflecting the current reality, given recent and/or periodic usage patterns, while at the same time improving scalability and reducing maintenance requirements.

2.7 Summary

The large amount of content in many online systems motivates the development of algorithms to aid users in browsing, searching and discovering new content from vast catalogs of items that may be interesting to them. Recommendation is especially useful for personalized content discovery. In this thesis we focus on the *top-N* recommendation task, or the *Find good items* task, according to the terminology used by Herlocker et al. in [Herlocker et al., 2004]. This task consists of finding the best N items for every user. We also distinguish between the types of user feedback used by recommendation algorithms. In the past, researchers have given great attention to ratings data, that contains explicit ratings – e.g. 1 to 5 star rating – given by users to items. More recently, an increasing amount of research has been done with positive-only data, typically containing implicit preferences of users – e.g. items bought, books read, movies watched. All our work in this thesis uses positive-only data. Another important aspect of real-world recommender systems is that they deal with time-evolving phenomena. User preferences change over time, new items and users enter, old ones leave, and cyclic temporal patterns – day vs night, weekdays vs weekends, summer vs winter – very often influence user preferences.

In this chapter, we have made an introduction to Collaborative Filtering (CF) techniques, specifically the ones most useful to our work. We have described classic neighborhood-based CF algorithms, Matrix Factorization (MF), and their capability to learn from positive-only data to perform top-N recommendations. We also have provided an overview to the most recent research on CF algorithms that explicitly deal with temporal dynamics. All the concepts and techniques introduced here are fundamental to understand the remainder of this thesis.

Chapter 3

Collaborative Filtering with streaming data

In real world systems, user feedback is continuously being generated, at unpredictable rates, and is potentially unbounded – in the sense that we can not assume it will ever end. In large scale systems, the rate at which user activity data is generated can be very fast. Building predictive models on these continuous flows of data is a problem actively studied in the field of data stream mining [Domingos and Hulten, 2000].

One efficient way to deal with data streams is to maintain incremental models and perform on-line updates as new data points become available. This simultaneously addresses the problem of learning non-stationary concepts and computational complexity issues. However, this requires algorithms able to process data at least as fast as it is generated. Incremental algorithms for recommendation are not frequently addressed in the recommender systems literature [Vinagre et al., 2014b; Matuszyk and Spiliopoulou, 2014]. In this chapter we describe the most important incremental algorithms for recommendation available and introduce a simple but fast incremental Matrix Factorization algorithm for positive-only feedback.

3.1 Data streams

Recent evolution of hardware and software has enabled the collection of large amounts of data. In the field of data mining, researchers try to find efficient methods and techniques to extract patterns and models from very large datasets [Hand et al., 2001]. In many cases, data is generated continuously, sometimes at a very fast rate, posing new challenges in storage, computation and data analysis capabilities [Domingos and Hulten, 2001]. Such

streams of information – data streams – do not have persistent relations, as new data keeps adding up at a potentially fast rate. Examples of data streams are network monitoring information, sensor network data, financial information, production line monitoring, web sites, among others [Babcock et al., 2002].

When mining data streams, the following differences from static datasets needs to be taken into consideration:

- Data elements arrive on-line;
- The system does not control neither the order at which elements arrive or the rate at which they are added;
- Data streams are potentially unbounded;
- Once a data element is processed it must be discarded or archived at some point – data streams typically do not fit in the available working memory –, meaning that subsequent accesses are harder or even impossible.

Looking at the data that recommender systems typically have to deal with, we verify that it shares all the characteristics of a data stream. User feedback is continuously generated online, at unpredictable rates and the length data is potentially unbounded. Having this in consideration, it becomes clear that the batch approach to recommender systems has fundamental limitations. The most obvious limitation is caused by the fact that the amount of user feedback data will never stop increasing. Even highly scalable algorithms may eventually become incapable of processing all the available data, which means some of the data will need to be discarded. One option is to "forget" past data, learning models over the most recent data. In [Vinagre and Jorge, 2012] we use forgetting mechanisms to forget past data, using sliding windows and fading factors. However, we find that simply forgetting past data is not beneficial in many cases, and can actually hurt the predictive ability of recommender systems. One other problem of batch algorithms is that, in order to keep up-to-date, they need to be retrained frequently. Between updates, users will hardly see relevant differences in recommendations, even though they have provided more information to the system meanwhile. In some applications, such as news recommendation, music streaming and automatic playlist generation, this can be an especially critical aspect of the system, and can severely affect the users' perception of the quality of the system.

P. Domingos and G. Hulten [Domingos and Hulten, 2001] identify the following desirable properties for a system that learns from data streams:

- It must process each data element faster that the rate of arrival of new elements;

- Main memory requirements must be bounded and independent of the number of data elements;
- Only a single pass over data should be necessary to build the model;
- The model must be available anytime, not only when it finishes processing the data, since data processing may never end;
- The model should be able to adapt to drifts in the underlying concept.

Stream mining algorithms should be able to timely process streams, at the risk of not being able to keep up with the arrival rate of data elements. In this thesis, we apply this principle to recommender systems. To achieve this, we look at the recommendation problem, specifically the top-N recommendation task, as a data stream problem. This approach has several implications in the algorithms' design – this Chapter and Chapter 4 – and evaluation – Chapter 5. Regarding the algorithms' design and implementation, one practical way to deal with data streams is to use algorithms that are able to update models incrementally. All algorithms studied in this thesis are incremental, in the sense that they are able to efficiently update the model using single data points arriving in the stream, without requiring access to past data.

3.2 Incremental neighborhood methods

Classic neighborhood-based CF algorithms – user- and item-based – have been adapted to work incrementally. The main idea in both cases is to maintain the factors of the similarity function in memory, and update them with simple increments each time a new user-item interaction occurs.

3.2.1 Incremental user-based CF

In [Papagelis et al., 2005], Papagelis et al. propose an algorithm that incrementally updates the values in the user-user similarity matrix. When a user u rates an item, the similarity values between u and other users are obtained with increments to previous values. Using the Pearson Correlation, the factors of the similarity calculation between user u and another user v are split in the following way:

$$A = \frac{B}{\sqrt{C}\sqrt{D}} \tag{3.1}$$

Given the set I of items co-rated by both u and v, factors A, B and C correspond to the following terms:

$A = sim(u, v)$,
$B = \sum_{i \in I}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)$,
$C = \sum_{i \in I}(r_{u,i} - \bar{r}_u)^2$,
$D = \sum_{i \in I}(r_{v,i} - \bar{r}_v)^2$

It is intended to obtain the new similarity A' from B, C and D, and the new incoming rating:

$$A' = \frac{B'}{\sqrt{C'}\sqrt{D'}} \Leftrightarrow A' = \frac{B + e}{\sqrt{C + f}\sqrt{D + g}} \tag{3.2}$$

where e, f and g are increments calculated after a new rating is available or an existing rating is updated.

To perform calculations, the values of B, C and D for all pairs of users must be available and need to be updated and stored each time new ratings arrive. Additionally, the average rating and the number of ratings for each user are also necessary to perform incremental calculations.

This simple technique allows fast online updates of the similarity values between the active user and all others.

3.2.2 Incremental user-based CF for positive-only feedback

Miranda and Jorge [Miranda and Jorge, 2009] study incremental user-based and item-based algorithms using positive-only user feedback.

The above formulation of the cosine in (2.10) for positive-only ratings allows the incremental calculation of item-item similarities based on user occurrence and co-occurrence counts. A user u is said to co-occur with user v for every item i they both occur with. An user-user co-occurrence matrix F containing the number of items common to each pair of users can be kept. The diagonal of F contains the number of independent occurrences of each user – i.e. the number of items the user occurs with. Every time a new user-item pair (u, i) is observed in the dataset, the corresponding counts are incrementally updated. Using these counts, the similarities of user u with any other user v can be easily recalculated and stored in a symmetric user-user similarity matrix S:

$$S_{uv} = \text{sim}(u, v) = \frac{F_{uv}}{\sqrt{F_{uu}} \times \sqrt{F_{vv}}} \tag{3.3}$$

Using (3.3), we implement Algorithm 3.1 for training, and Algorithm 3.2 to produce a recommendation list for a user u.

Algorithm 3.1: UKNN - User-based incremental algorithm (training)

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
Data: R the user-item ratings matrix
output: S the user-user similarity matrix

for $(u,i) \in D$ **do**
  for $v \in \{x | R_{xi} = 1\}$ **do**
    $F_{uv} \leftarrow F_{uv} + 1$
    $F_{uu} \leftarrow F_{uu} + 1$
    $S_{uv} \leftarrow \frac{F_{uv}}{\sqrt{F_{uu}} \times \sqrt{F_{vv}}}$
  $R_{ui} \leftarrow 1$

Algorithm 3.2: UKNN - User-based incremental algorithm (recommendation)

Data: S the user-user similarity matrix
Data: R the user-item ratings matrix
input : u the user
input : k the number of neighbors
input : n the number of recommendations
output: A recommendation list of length n

init:
  $K_u \leftarrow \texttt{FindKNN}(u,k)$
  $I_u \leftarrow \{x | R_{ux} = 0\}$
for $i \in I_u$ **do**
  $score_i \leftarrow \frac{\sum_{v \in K_u} S_{uv} R_{vi}}{\sum_{v \in K_u} S_{uv}}$
$rec_u \leftarrow \texttt{SortByScore}(I_u)$
return $\texttt{Truncate}(rec_u, n)$

3.2.3 Incremental item-based collaborative filtering

Alternatively to the incremental user-based algorithm we can use the incremental approach in item-based algorithms. Incremental item-based CF is trivially obtained using the exact same formulation of the user-based algorithm. One practical way of looking at it is simply to transpose the user-item ratings matrix and then apply the exact same algorithm. The result will be an incrementally maintained item-item similarity matrix.

3.2.4 Limitations of neighborhood-based incremental CF

One problem with incremental similarity computation is that it requires the calculation of the k-nearest neighbors of all items after the similarities are updates. The algorithms in [Papagelis et al., 2005] and [Miranda and Jorge, 2008] do neighborhood search at recommendation time, which can slow down the response time of the recommender system. In the batch algorithm, one obvious way to avoid this is to pre-compute all neighborhoods at the training stage. However, when using incremental algorithms, this would mean that neighborhood computations would have to be performed after every incremental step, severely hurting the update time for each observation. The decision of in which step – training or recommendation – should neighborhoods be computed is therefore dependent on what works best for the practical application.

In the experimental work in chapter 5 we use an incremental neighborhood-based algorithm that computes neighborhoods at the incremental training step – i.e. every time similarity values are updated.

3.3 Incremental matrix factorization

Early work on incremental matrix factorization for recommender systems is presented in [Sarwar et al., 2000b], where Sarwar et al. propose a method to perform incremental updates of the Singular Value Decomposition (SVD) of the ratings matrix. This is a direct application of the Fold-in method [Deerwester et al., 1990], that essentially enables the calculation of new latent vectors (corresponding to new users or new items) based on the current decomposition and by appending them to the corresponding matrices. One shortcoming of this method is that it is applicable only to pure SVD, and it requires initial batch-trained model with the problems mentioned in Section 2.3.3.

Most matrix factorization algorithms for recommendation are typically seen as batch, iterative procedures. This is the natural approach, given that the most successful techniques

in the field involve iterative methods such as Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS). In this section we review the literature on incremental matrix factorization for recommendation and we propose a simple SGD algorithm can be used as a incremental process with positive-only data. The algorithm updates factor matrices every time a new positive user-item interaction arrives in the stream of user feedback.

3.3.1 Incremental BRISMF

Takács et al. address the problem of incremental model updates in [Takács et al., 2009]. The idea is to retrain user features every time new ratings are available, but *only* for the active user(s), leaving item features unmodified, avoiding the whole process of batch-retraining the model. The authors first train the factorized model using their algorithm BRISMF, which is essentially equivalent to BSGD (Algorithm 2.1), with the addition of user and item biases. User biases try to capture how users tend to use the rating scale – some users tend to give higher ratings than others. Item biases account for how each item tends to be rated – some receive higher ratings than others. Biases are trained jointly with feature vectors. By leaving biases out, BRISMF is equivalent to BSGD. The algorithm proposed by Takács et al. to incrementally maintain a model is described in Algorithm 3.3. For the sake of simplicity, we leave the biases out.

Algorithm 3.3: Incremental user feature updates

$\mathbf{Data}$: a finite set or a data stream $D = \{(u,i,r)_1, (u,i,r)_2, \ldots\}$
$\mathbf{input}$: $iter$ the no. of iterations
$\mathbf{input}$: λ the regularization factor
$\mathbf{input}$: η the learn rate
$\mathbf{input}$: R the ratings matrix
$\mathbf{input}$: A the user factor matrix
$\mathbf{input}$: B the item factor matrix
$\mathbf{output}$: A the updated user factor matrix

1 **for** $(u,i,r) \in D$ **do**
2 $R_{ui} \leftarrow r$
3 $A_u \sim \mathcal{N}(0, 0.1)$
4 **for** $k \leftarrow 1$ **to** $iter$ **do**
5 **for** $\{j | \exists_{R_{uj}}\}$ **do**
6 $err_{uj} \leftarrow R_{uj} - A_u \cdot B_j$
7 $A_u \leftarrow A_u + \eta(err_{uj} B_j - \lambda A_u)$

While Algorithm 3.3 may be enough to solve many real world problems, there is also a number of important limitations:

1. The algorithm requires A and B – the user and item feature matrices – to be pre-computed with the batch algorithm;
2. The whole ratings history R is required at the core of the algorithm;
3. Item features are not updated, and new items are not accounted for.

These limitations mean that the incremental process proposed by Takács is helpful, but does not solve the whole problem. The batch algorithm is still necessary for initial model building and to incorporate item updates from time to time. Moreover, the model requires the whole history to be available at all times.

3.3.2 Incremental learn-to-rank

Learning to rank [Liu, 2011] encompasses a set of methods that use machine learning to model the precedence of some entities over others, assuming that there is at natural ordering between them. The top-N recommendation task consist of retrieving the best ranked items for a particular user, so it is natural to approach the task as a learn-to-rank problem. Moreover, learning-to-rank algorithms can be trained using factorization models and SGD [Burges et al., 2005], which fits nicely into the recommender systems framework.

This is the approached followed by Rendle et al. in [Rendle et al., 2009] with their Bayesian Personalized Ranking (BPR) framework, that we describe in Chapter 2, Section 2.3.5, specifically focusing on the BPRMF algorithm (Algorithm 2.3). One shortcoming of this algorithm is that it is approached as a batch method. However, although not documented in the literature, the algorithm can easily work incrementally, as we show below in Sec. 3.3.2.1.

Another incremental algorithm for ranking that uses a selective sampling strategy is proposed by Diaz-Aviles et al. in [Diaz-Aviles et al., 2012]. The algorithm maintains a reservoir with a fixed number of observations taken randomly from a stream of positive-only user-item pairs. Every n^{th} pair in the stream is sampled to the reservoir with probability $|R|/n$, with $|R|$ being the number of examples in the reservoir. Model updates are performed by iterating through this reservoir rather than the entire dataset. At each iteration, one user-item pair is randomly selected from the reservoir, and then the authors use the “59 trick” [Smola and Schölkopf, 2000] to sample non-observed items for the user in the selected pair. The trick consists of sampling 59 instances, and guarantees that with probability of 0.95, at least 1 instance of the 59 will be within the best 5% estimates. Then an item from the 59 is sampled with probability proportional to its “informativeness”. The most informative items are the ones with opposite labels, but close together in the ranking. By consecutively sampling and adjusting these pairwise ranks, the global model eventually converges to an

optimum. The rationale is that the training always uses the most informative examples to update the model.

Two incremental methods using Stochastic Gradient Descent (SGD) are evaluated in [Ling et al., 2012]. Our work differs from this for two fundamental reasons: first, we are using positive-only data and second, we are dealing with a data stream of user feedback. For both reasons, our entire framework, including implementation and evaluation methodology (see Chapter 5) is significantly different.

3.3.2.1 Incremental BPRMF

In [Rendle et al., 2009], Rendle et al. use BPRMF in a stationary setting, and do not provide information relative to if and how it is possible to use BPRMF incrementally. Although not documented in the literature, the implementation available in the MyMediaLite[1] software library [Gantner et al., 2011] provides a fully functional implementation of an incremental version of the algorithm. The pseudo-code of this version is available in Algorithm 3.4. Essentially, instead of iterating over a whole dataset in batch mode, the algorithm iterates over user-item pairs one or more at a time, as they become available in the user feedback stream.

Similarly to batch BPRMF (Algorithm 2.3), the sampling performed by the function `SampleFrom` can be done uniformly or to sample with probability proportional to the popularity of items (WBPRF). As with the batch version, $\sigma(x) = \frac{e^{-x}}{1-e^{-x}}$. In Chapter 5, we use both BPRMF and WBPRMF in our experiments.

3.4 Forgetting

One of the problems of learning from data streams is that the concepts being learned are typically not static. In recommender systems, it has been shown that users change their opinion about some items, over time [Koychev, 2000; Koren, 2009]. This means that an algorithm that correctly models user preferences in a certain point in time does not accurately represent the same users' preferences some time later. Incremental algorithms benefit from being constantly updated with fresh data, therefore capturing these changes immediately, however the model still retains the concepts learned from past data. One way to deal with this is to forget this outdated information, i.e. data that does no longer represent the concept(s) being learned by the algorithm.

[1] http://www.mymedialite.net/

Algorithm 3.4: BPRMF - incremental version

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ_u, λ_i and λ_j the regularization factors
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

for $(u,i) \in D$ **do**
  if $u \notin \texttt{Rows}(A)$ **then**
    $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
    $A_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin \texttt{Rows}(B)$ **then**
    $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
    $B_i \sim \mathcal{N}(0, 0.1)$
  for $count \leftarrow 1$ **to** $iters$ **do**
    $j \leftarrow \texttt{SampleFrom}(\{j | (u,j) \notin D\})$
    $A_u \leftarrow A_u + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)(B_i - B_j) + \lambda_u A_u)$
    $B_i \leftarrow B_i + \eta(\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_i B_i)$
    $B_j \leftarrow B_j + \eta(-\sigma(A_u \cdot B_i - A_u \cdot B_j)A_u + \lambda_j B_j)$

3.4.1 Forgetting for neighborhood-based incremental CF

In our past work [Vinagre and Jorge, 2012] we have used fading factors to *gradually* forget user feedback data using neighborhood-based algorithms. We do this by successively multiplying by a positive scalar factor $\alpha < 1$ all cosine similarities between all pairs of users – or items, in an item-based algorithm – at each incremental step, before updating the similarities with the new observations. If we consider a symmetric similarity matrix S containing all similarity values between pairs of users – or pairs of items –, this is achieved using the update $S \leftarrow \alpha S$. The lower the value of α, the faster the forgetting occurs. In practice, two users – or two items – become less similar as they co-occur farther apart in time.

Our results show that this type of forgetting is beneficial for the algorithms' accuracy, especially in the presence of sudden changes.

3.4.2 Forgetting with factorization-based incremental CF

During the work on this thesis, we have also investigated forgetting strategies for incremental matrix factorization algorithms, in a collaboration with Pawel Matuszyk and Myra Spiliopoulou [Matuszyk et al., 2015]. To achieve forgetting we use a total of eleven forgetting strategies of two types: *rating-based* and *latent-factor-based*. The first performs forgetting of certain past ratings for each user, while the latter performs forgetting by readjusting the latent factors in the user factor matrix, diminishing the impact of past ratings.

Ratings-based forgetting generally consists of forgetting sets of ratings. Formally, it is a function that operates on the set of ratings R_u of a user u and generating a new set $R'_u \subseteq R_u$:

$$f : R_u \rightarrow R'_u$$

Matuszyk, in collaboration with us, proposed the following six rating-based forgetting methods:

- *Sensitivity-based forgetting*, based on sensitivity analysis. The idea is to forget the ratings that cause changes with higher-than-normal magnitude in the user model. The rationale is that these ratings are typically not representative of the user preferences and should therefore be forgotten. Practical examples of such ratings are the ones on items that are bought as gifts, or when some person uses someone else's account. If for some reason these outliers become more frequent – e.g. the user has actually changed preferences –, the model automatically begins accepting them after some short amount of time.

- *Global sensitivity-based forgetting*. Like the previous technique, it also forgets ratings that have an impact that falls out of the regular one. The difference is that the sensitivity threshold is measured globally instead of being personalized.

- *Last N retention*. Here the strategy is to retain the latest N ratings for each user. This acts as a sliding window over the ratings of each user with at most N ratings.

- *Recent N retention*. Similar to Last N retention, except that N corresponds to time, instead of a rating count, implementing a variable size time-based window, retaining only the ratings that fall into the previous N time units. In practice, it attributes a lifetime of N to each rating.

- *Recall-based change detection*. This strategy detects sudden drops in the incremental measurement of Recall – i.e. downward variations above a certain threshold, which is maintained incrementally as well – and forgets all ratings occurring before the detected change. This is particularly helpful in environments where changes are relatively abrupt and benefit from completely resetting user profiles.

- *Sensitivity-based change detection*[2]. This is similar to Recall-based change detection, except that the criterion for detecting a change is the impact of new ratings. If a certain rating changes the user profile dramatically, we assume that the change is real – the user has actually changed preferences – and forget all past ratings.

Latent-factor-based forgetting operates directly on the factorization model, adjusting user or item latent factors in a way that it imposes some type of forgetting. These adjustments to latent factors are linear transformations in the form:

$$A_u^t = \gamma \cdot A_u^{t+1} + \beta$$

$$B_i^t = \gamma \cdot B_i^{t+1} + \beta$$

In the above equations, γ and β are dependent on one of the five strategies below:

- *Forget unpopular items*. This technique consists of penalizing unpopular items by multiplying their latent vectors with a factor proportional to their frequency in the stream.

- *User factor fading*. Here, user latent factors are multiplied by a positive factor $\gamma < 1$. This causes the algorithm to gradually forget user profiles, benefiting recent user activity and penalizing past activity.

[2]The term *sensitivity* is used here with its broader meaning, not as a synonym of recall.

- *SD-based user factor fading*. This technique also multiplies user factors by a scalar value, except that this value is not a constant, but rather depends on the volatility of user factors. Users whose factors are more unstable have a higher forgetting rate than those whose profiles are more stable.
- *Recall-based user factor fading*. Similarly to the previous strategy, users have differentiated forgetting factors. This technique amplifies the forgetting factor for users that have low Recall.
- *Forget popular items*. This is the opposite of "Forget unpopular items". Frequent items are penalized as opposed to the non-frequent ones.

Using Algorithm 3.3 as baseline, we implement and evaluate the above strategies on eight datasets, four of which contain positive-only data, while the other four contain numerical ratings.

Our findings in [Matuszyk et al., 2015] show that forgetting significantly improves the performance of recommendations in both types of data – positive-only and ratings. Latent-factor-based forgetting techniques, and particularly "SD-based user factor fading", have shown to be the most successful ones both on the improvement of recommendations and in terms of computational complexity.

3.5 Summary

In real-world systems, usage data keeps adding up as new users and items enter the system and new ratings are provided. It is a fundamental requirement that CF algorithms are scalable enough to cope with these increasing amounts of data. In this thesis we give great importance to scalability issues, namely the ability to deal with natural data at the rate that it is generated online, is crucial. Having this in mind, one natural approach to recommendation data is to process it as a data stream. This approach enables us to simultaneously capture time dynamics and require less computational resources.

User feedback data has a dynamic nature. It is continuously being generated at potentially fast rates, and it is ever growing. Traditional approaches to recommender systems treat data in batch and models typically need to be rebuilt from scratch when they become out-of-date. This is an important limitation of batch approaches. In this chapter, we have presented our rationale for approaching user feedback data as a *data stream* and have studied existing incremental algorithms and strategies for recommendation with streaming data. One of these strategies is to forget outdated or non-representative information. We have studied several forgetting strategies in incremental in the past, and also very recently,

in collaboration with other researchers. Our results show that incremental recommendation algorithms clearly benefit from forgetting past data.

Chapter 4

Stream-based recommendations with negative feedback

In the previous chapter we have described a number of algorithms and techniques that are especially suitable for learning recommendation models from a data stream of user feedback. As stated in Chapter 1, we are particularly interested in exploiting positive-only data, simply because it is a more common application scenario. In this chapter we present two main contributions. First, we propose an incremental matrix factorization algorithm for streams of positive-only user feedback. This algorithm – ISGD – outperforms both state-of-the-art learning-to-rank algorithms [Rendle et al., 2009] and classic neighborhood-based algorithms [Miranda and Jorge, 2008] for recommendation.

The second contribution is a recency-based technique to deal with positive-only feedback. Recommendation algorithms that work with positive-only data essentially have to distinguish between good and bad recommendations for each user, but training only on the *good* examples – i.e. negative feedback is absent. This problem is also known as One-Class Collaborative Filtering (OCCF) [Pan et al., 2008; Paquet and Koenigstein, 2013], given its similarity to One-class Classification [Khan and Madden, 2014]. One-class Classification is the most extreme instance of class imbalance in a two-class problem, in which all available examples for training belong to the same class. Naively using conventional classification algorithms in such problems can easily result in either over-generalization or overfitting. In collaborative filtering algorithms, these problems are amplified by the need to capture several concepts – one for each user – in a single model [Pan et al., 2008].

4.1 Proposed algorithm - ISGD

The first contribution of this thesis consists of an incremental matrix factorization algorithm for recommendation. The algorithm is intended to work with positive-only feedback, and is fundamentally memoryless, since it does not require access to past data.

The optimization process of Algorithm 2.1 consists of a batch process, given that it requires several passes – iterations – through a learning dataset to train a model. While this may be an acceptable overhead in a static environment, it is not acceptable for streaming data. As the number of observations increases and is potentially unbounded, repeatedly revisiting all available data eventually becomes too expensive to be performed online.

Fortunately, SGD is *not* a batch algorithm, as is made quite clear by Le Cun et al. in [LeCun et al., 1996]. The only reason why several passes are made over a (repeatedly shuffled) set of data is because there is a *finite* number of examples. Iterating over the examples in different order several times is basically a trick to improve learning process in the absence of fresh examples. If we assume – as we have to, in a data stream scenario – that there is a continuous flow of examples, this trick is no longer necessary. By this reasoning, SGD can – and should – be used online. This idea is also emphasized by Bottou in [Bottou, 2003].

In [Vinagre et al., 2014b], we propose Algorithm 4.1, designed to work as an online process, that updates factor matrices A and B based solely on the current observation. This algorithm, despite its formal similarity with Algorithm 2.1, has two practical differences. First, the learning process requires a single pass over the available data – i.e. there is no need to revisit past observations. Note that in Algorithm 4.1, at each observation (u, i), the adjustments to factor matrices A and B are made in a single iteration. Second, no data shuffling – or any other data pre-processing – is performed. Given that we are dealing with positive-only feedback we approach the boolean matrix R by assuming the numerical value 1 for $true$ values. Accordingly, we measure the error as $err_{ui} = 1 - \hat{R}_{ui}$, and update the rows in A and B^T using the update operations in (2.14). We refer to this algorithm as ISGD.

Since we are mainly interested in top-N recommendation, we need to retrieve an ordered list of items for each user. We do this by sorting candidate items i for each user u using the function $f_{ui} = |1 - \hat{R}_{ui}|$, where $\hat{R}_{ui}$ is the non-boolean predicted score. In plain text, we order candidate items by descending proximity to value 1.

In [Vinagre et al., 2014b] we show that this algorithm outperforms other state-of-the-art incremental factorization algorithms in most cases and with the best runtime performance in all cases. In this thesis, we use Algorithm 4.2 – ISGD –, a slightly different version of Algorithm 4.1. The only difference between ISGD-SI and ISGD is that the first iterates only once over each observation, while the latter iterates $iter$ times – ISGD-SI is ISGD with $iter$ set to 1. Evidently, there is an additional cost for iterating more than once. However, we

Algorithm 4.1: ISGD-SI - Incremental SGD for positive-only ratings with single iteration

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : λ the regularization factor
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

1 **for** $(u,i) \in D$ **do**
2 **if** $u \notin \texttt{Rows}(A)$ **then**
3 $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
4 $A_u \sim \mathcal{N}(0, 0.1)$
5 **if** $i \notin \texttt{Rows}(B)$ **then**
6 $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
7 $B_i \sim \mathcal{N}(0, 0.1)$
8 $err_{ui} \leftarrow 1 - A_u \cdot B_i$
9 $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
10 $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$

show in chapter 5 that the cost of adding iterations is very low.

4.1.1 Stream-based bagging

Bagging [Breiman, 1996] – or bootstrap aggregating – is an ensemble technique that takes a number of bootstrap samples of a dataset and then trains a model on each one of the samples. Then predictions from the various sub-models are aggregated in a final prediction. The objective of bagging is to deal with the instability of some algorithms that are overly sensitive to small changes in the data. In short, bagging aims at reducing variance. The diversity offered by training several models with slightly different bootstrap samples of the data helps in giving more importance to the main concepts being learned – since they must be present in most bootstrap samples of the data – , and less importance to noise or irrelevant phenomena that may mislead the learning algorithm.

To obtain a bootstrap sample of a dataset with size N, we perform N trials, sampling a random example with replacement from the dataset. Each example has probability of $1/N$ to be sampled at each trial. The resulting dataset will have the same size of the original dataset, however some examples will not be present whereas some others will occur multiple times, as a result of sampling with replacement. To obtain M samples, we simply repeat the process M times.

Algorithm 4.2: ISGD - Incremental SGD for positive-only ratings, with multiple iterations

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
output: A the user factor matrix
output: B the item factor matrix

1 **for** $(u,i) \in D$ **do**
2 **if** $u \notin \texttt{Rows}(A)$ **then**
3 $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
4 $A_u \sim \mathcal{N}(0, 0.1)$
5 **if** $i \notin \texttt{Rows}(B)$ **then**
6 $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
7 $B_i \sim \mathcal{N}(0, 0.1)$
8 **for** $n \leftarrow 1$ **to** $iter$ **do**
9 $err_{ui} \leftarrow 1 - A_u \cdot B_i$
10 $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
11 $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$

In the recommender systems field, bagging has been seldom used. In [Segrera and Moreno, 2006] experiments bagging and boosting [Freund and Schapire, 1996] – another ensemble method – with content-based recommender systems. Results show that a decision tree algorithm benefits considerably from using ensemble techniques, especially with bagging. In [Jahrer et al., 2010], bagging is used in conjunction with a boosting algorithm with significant improvements in accuracy. In both contributions, this is done in batch.

As originally described in [Breiman, 1996] for classification and regression problems, bagging is a batch procedure that requires $N \times M$ passes through the dataset. However, Oza and Russel have shown [Oza and Russell, 2001] that this can be done in a single pass if the number of examples is very large ($N \to \infty$), which is common when learning from data streams. In the batch method above, each bootstrap sample contains K occurrences of each example, with $K \in \{0, 1, 2, \ldots\}$, and:

$$P(K = k) = \binom{N}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{N-k} \tag{4.1}$$

In a streaming setting, one can just use the equation above to take every new example and training it K times in each sample. However, this still requires knowing the size of the dataset N beforehand. Oza and Russel show that with $N \to \infty$, the distribution of K tends to a $Poisson(1)$ distribution, and therefore

$$P(K = k) = \frac{e^{-1}}{k!} \tag{4.2}$$

which eliminates the need of any prior knowledge about the data.

We propose the application of online bagging in ISGD (Algorithm 4.2). The resulting algorithm – BaggedISGD - Algorithm 4.3 – can improve its performance in all datasets (see Section 5.6). There is an overhead, since the algorithm requires at least M times the computational resources needed for ISGD, with M bootstrap nodes.

Algorithm 4.3 learns a model based on M submodels. To perform the actual list of recommendations for a user u, items i are sorted by a function $f = |1 - \hat{R}_{ui}|$ as with ISGD. The scores $\hat{R}_{ui}$ are the average score of all submodels:

$$\hat{R}_{ui} = \frac{\sum_{m=1}^{M} \hat{R}_{ui}^{m}}{M} \tag{4.3}$$

To our best knowledge, this is the first online ensemble method for recommender systems contribution in the literature. We evaluate bagging in Section 5.6.

Algorithm 4.3: BaggedISGD - Bagging version of ISGD (training)

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
input : M the number of bootstrap nodes
output: the set of M user factor matrices A^m
output: the set of M item factor matrices B^m

for $(u,i) \in D$ **do**
  for $m \leftarrow 1$ **to** M **do**
    $k \sim \text{Poisson}(1)$
    if $k > 0$ **then**
      for $l \leftarrow 1$ **to** k **do**
        if $u \notin \texttt{Rows}(A^m)$ **then**
          $A_u^m \leftarrow \texttt{Vector}(\text{size} : k)$
          $A_u^m \sim \mathcal{N}(0, 0.1)$
        if $i \notin \texttt{Rows}(B^m)$ **then**
          $B_i^m \leftarrow \texttt{Vector}(\text{size} : k)$
          $B_i^m \sim \mathcal{N}(0, 0.1)$
        for $n \leftarrow 1$ **to** $iter$ **do**
          $err_{ui} \leftarrow 1 - A_u^m \cdot B_i^m$
          $A_u^m \leftarrow A_u^m + \eta(err_{ui} B_i^m - \lambda A_u^m)$
          $B_i^m \leftarrow B_i^m + \eta(err_{ui} A_u^m - \lambda B_i^m)$

4.2 Negative preference imputation

One particular challenge of recommendation with positive-only data is how to interpret *absent* user-item interactions. These can be seen as either negative or unknown preferences. For example, consider a user that navigates through a large collection of movies on an online movie streaming service. Every time she decides to watch a movie, this decision is recorded. For every user, the system maintains a record containing all the user's rental history and thus providing valuable information about that user's preferences. However it does not contain any information about which movies the user does *not* like. Among all the movies the user did not watch, are the ones that she does not like *and* the ones she does not know. Given that in our problem setting we need to learn a predictive model by analysing positive-only user-item preferences, one important part of the problem is how to distinguish between the two possible interpretations of *absent* user-item interactions in the feedback data. If an interaction between a user and an item does not occur, this can either be interpreted as a candidate preference – the user does not know the item – or as negative preference – the user does not like the item. Ideally, the recommender should consider the item for recommendation in the first case and exclude it in the second. However, it is not trivial to make a clear distinction between negative and candidate items when only positive examples are available.

One possible approach to positive-only data is to formulate recommendation as a learn-to-rank problem [Rendle et al., 2009], which directly models user preferences as a ranked set. Another approach is to perform artificial imputation of negative examples. In the literature we are able to find two main schemes to perform negative example imputation: sampling and weighting [Pan et al., 2008]. Both are based on the intuition that users with high activity level – users that interact with many items – are more likely to dislike the items that they do not interact with. Conversely, users with little activity are more likely to have preferences among the items for which they do not express a positive interaction.

4.2.1 Weighting methods

The two extreme ways to look at unobserved user-item pairs in positive-only data is to (a) consider them as all negative examples, meaning that users only like what they interact with and nothing else or (b) consider them all unknown examples, assuming that all of them are potential recommendations. In a typical application, none of the extremes is a realistic scenario, because unobserved interactions may fall in either one of the two cases – the user does not like or the users does no know. In [Pan et al., 2008], Pan et al. use the intuition that the likeliness of an unobserved user item pair in the data is related to the activity rate of users and items. The intuition is the following: very active users tend to

cover the item space more broadly, decreasing the chances of leaving out possible good recommendations. Additionally, items with many positive interactions are more likely to be good recommendations to any user.

Using this intuition in a matrix factorization problem can be easily done by adding confidence levels to the user-item matrix R, that contains the values 1 or 0 depending on whether a positive interaction is observed or not for every user-item pair. The confidence levels can be stored in an additional matrix W using the following scheme:

$$\begin{aligned} R_{ui} = 1 &\Rightarrow W_{ui} = 1 \\ R_{ui} = 0 &\Rightarrow W_{ui} \in [0, 1] \end{aligned} \tag{4.4}$$

With $R_{ui} = 0$ and as $W_{ui} \to 1$, this can be interpreted as (u, i) being more likely to be a negative example. The factorization problem first formalized in (2.13) then becomes:

$$\min_{A.,B.} \sum_{(u,i)\in R} W_{ui}((R_{ui} - A_u \cdot B_i)^2 + \lambda(||A_u||^2 + ||B_i||^2)) \tag{4.5}$$

The problem is now how to fill in values in W for unobserved user-item pairs, according to the second line of (4.4). Pan et al. use three methods:

1. Uniform weighting: assign a random value $\delta \in [0, 1]$ uniformly
2. User-based weighting: assign a value by user proportional to the the number of observations for that user $W_u \propto \sum_i R_{ui}$
3. Item-based weighting: assign a value by item proportional to the number of users that do not interact with that item $W_i \propto \sum_u (1 - R_{ui})$

Methods 2. and 3. use the intuition above. Missing pairs are more likely to be considered negative if the user is very active or the item has low activity. Based on the original dataset D, consisting of (u, i) user-item pairs, a hyperparameter w can be used to control the global relative weight of positive vs. negative examples:

$$w = \frac{\sum_{(u,i)\notin D} W_{ui}}{|D|} \tag{4.6}$$

To obtain the actual weights W_{ui} according to methods 2. and 3., the algorithm needs to analyze the whole dataset before learning the model.

This is the same approach followed by Hu et al. in [Hu et al., 2008], that also suggest the use of this technique in a practical application of a Video-On-Demand service, in which

the weights are obtained directly from the time users spend watching a video item. For instance, if a user watches a particular item for its total duration, the weight for that user-item interaction would be $W_{ui} = 1$. The authors also provide a technique that helps in explaining recommendations to users, a feature that is known to help increasing the trust in the system [van Rijn et al., 2014].

Note that in (4.5) the sum needs to be performed over all user-item pairs – observed or not – for which $W_{ui} \neq 0$. Using either one of the three methods above it becomes evident that the vast majority of W values are non-zero. This means that the algorithm needs to iterate over practically all user-item combinations, regardless if they were actually observed. This bears a potentially huge cost, since typically the actually observed positive user-item pairs are a tiny portion of the possible combinations. An alternative scheme, that alleviates this cost is to use a sampling strategy.

4.2.2 Sampling methods

One other approach proposed by Pan et al. in [Pan et al., 2008] samples over the unobserved user-item interactions – the empty values in the user-item matrix – to introduce them as *negative* user-item interactions in the original data, using a pre-defined sample size q as a user-defined parameter. There are three strategies to sample user-item pairs (u, i) from the set of unobserved interactions R_0:

1. Sample uniformly: all unobserved pairs are sampled with the same probability;

2. Sample user-by-user proportionally to the user activity rate;

3. Sample item-by-item proportionally to the inverse item activity rate.

One obvious advantage of the sampling method, especially for large scale problems, is the ability to define – and limit – in advance how much more data the algorithm needs to handle, compared to the size of the original dataset. The amount of additional data points – the sampled negative (u, i) pairs – to process is a user defined hyperparameter q. Similarly to the parameter w in the weighting scheme above, Pan et al. define q as a proportion of negative examples to sample relative to the number of examples in the original dataset.

Similarly to weighting methods, methods 2. and 3. require the previous analysis of the entire dataset to obtain the user and item activity rates.

4.2.3 Graphical models

In [Paquet and Koenigstein, 2013], Paquet and Koenigstein use a graph-based approach to infer likely negative preferences of users. The technique consists of trying to infer a subgraph of negative user-item interactions maintaining the degree distributions of the original graph of positive-only interactions. Like the two previous methods, this requires the batch pre-processing of the data to obtain the necessary statistics or data dependent parameters.

4.3 Recency-based negative feedback

As mentioned in Section 4.1, the positive class in ISGD-SI and ISGD (Algorithms 4.1 and 4.2) is encoded as the rating value 1. One problem of this approach is that the absence of negative examples leads to a model that converges globally to the positive class. This causes predictions to accumulate closer and closer around the target positive value. Eventually, the algorithm looses discriminative power, causing accuracy degradation. Figure 4.1 illustrates the phenomenon. This figure is produced using the evaluation methodology described in Chapter 5. The line is drawn using a moving average of Recall, and represents the evolution of this metric as the algorithm learns from data. For this particular illustration, we have replicated the YHM-6KU dataset (see Table 5.1) three times to make it long enough to reveal the phenomenon, and then shuffled the resulting replicated dataset to eliminate any time-related dynamics. The accuracy of ISGD steadily degrades over time. This is a consequence of using an algorithm that is originally designed for ordinal ratings, which essentially approaches the problem as a regression task, and obviously does not account for the absence of negative examples.

To solve this problem we propose two recency-based mechanisms to select likely negative examples. The intuition is that the items that have occurred the longest ago in the data stream are better candidates to be taken as negative examples for any user. These are items that no users have interacted with in the longest possible period of activity in the system. We address the problem using two approaches. Both consist of maintaining a global priority queue of all items items occurring in the stream – independently of the user. For every new positive (u, i) in the data stream, we introduce a set $\{(u, j_1), \ldots, (u, j_l)\}$ of negative feedback consisting of the active – currently observed – user u and the l items j that are in the tail of the global item queue. The two approaches differ on the criteria used to maintain the order – i.e. the priorities of items – of the queue.

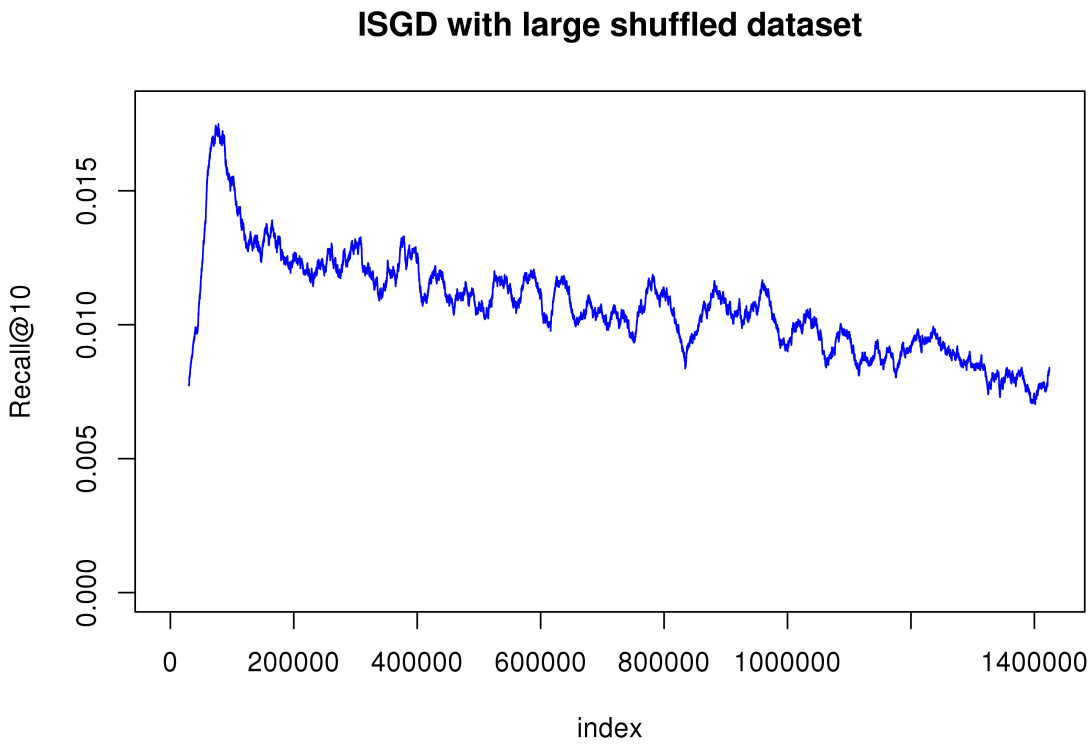


Figure 4.1: Evolution of Recal@10 of ISGD through the incremental learning and prediction process, illustrating the degradation of ISGD over time.

4.3.1 Recency-based algorithm

Our first approach prioritizes the most recent items over the least recent ones. The set of the l items in the tail of queue – the ones that occurred the farther back in the data stream – are selected for negative feedback. We apply this scheme in ISGD (Algorithm 4.2) using a FIFO (First-In-First-Out) queue globally – common to all users –, containing all items seen so far in the stream. Every time an item occurs in the stream, it is moved to – or inserted at, if new – the head of the queue. A set of items in the tail of the queue are selected for negative feedback. To avoid penalizing infrequent items repeatedly, every time an item is used as negative feedback it is also moved to the head of the queue. The process is illustrated in Figure 4.2, for the simplest case when the negative feedback amount is $l = 1$.

In Algorithm 4.4 the model correction with negative feedback is done by measuring error with respect to the negative class 0 – instead of 1 –, associated with a set of items in the tail of the FIFO queue and the active user. The length of this set is given by the user defined parameter l. The queue related functions `initqueue()`, `enqueue()`, `dequeue()` and `remove()` respectively perform queue initialization, head insertion, tail removal, and

Algorithm 4.4: RAISGD: Recency-Adjusted ISGD

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
input : l the number of negative examples
output: A the user factor matrix
output: B the item factor matrix

$Q \leftarrow \texttt{initqueue}()$
for $(u,i) \in D$ **do**
  if $u \notin \texttt{Rows}(A)$ **then**
    $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
    $A_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin \texttt{Rows}(B)$ **then**
    $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
    $B_i \sim \mathcal{N}(0, 0.1)$
  for $k \leftarrow 1$ **to** $\texttt{min}(l, \#Q)$ **do**
    $j \leftarrow \texttt{dequeue}(Q)$
    for $k \leftarrow 1$ **to** $iter$ **do**
      $err_{uj} \leftarrow 0 - A_u \cdot B_j$
      $A_u \leftarrow A_u + \eta(err_{uj} B_j - \lambda A_u)$
    $\texttt{enqueue}(Q, j)$
  for $k \leftarrow 1$ **to** $iter$ **do**
    $err_{ui} \leftarrow 1 - A_u \cdot B_i$
    $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$
  if $i \in Q$ **then**
    $\texttt{remove}(Q, i)$
  $\texttt{enqueue}(Q, i)$

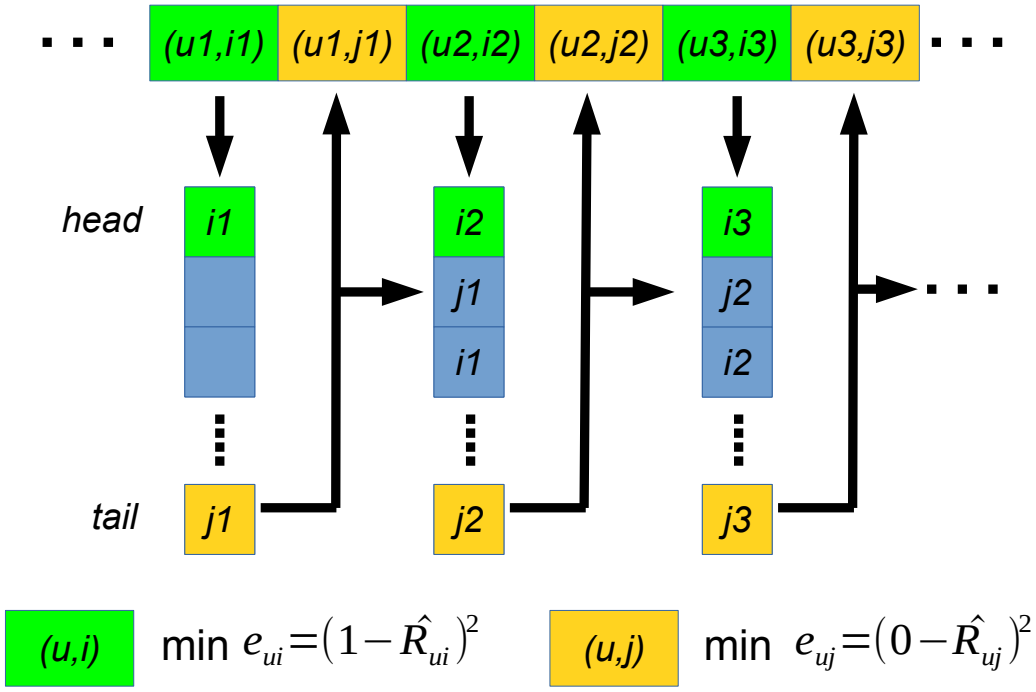


Figure 4.2: Recency-based negative feedback imputation: an item queue, common to all users, ordered by recency is maintained alongside the model. For each (green) positive (u, i) pair in the stream (top), we insert one or more negative feedback pairs (yellow) for that user, combined with the least-recent item(s) in the tail of the queue. We update the model minimizing the error with respect to value 1 for the positive feedback and with respect to value 0 for the negative feedback. All items used in the update (both positive and negative) are moved to the head of the queue.

index-based removal. The update operations corresponding to negative feedback only change the user factor matrix, leaving the item factor matrix unmodified. We have verified empirically that updating item features with negative feedback is actually harmful to the model's predictive ability. This is possibly explained by the intrinsic stability of items [Takács et al., 2009].

This method is able to deal with the degradation shown above in Figure 4.1. A comparison with ISGD is shown in Figure 4.3. In this example, RAISGD and ISGD were given the exact

same dataset and the same hyperparameters k, $iter$, η, λ. RAISGD's negative feedback amount was set with $l = 1$. Clearly, the problem does not affect RAISGD.

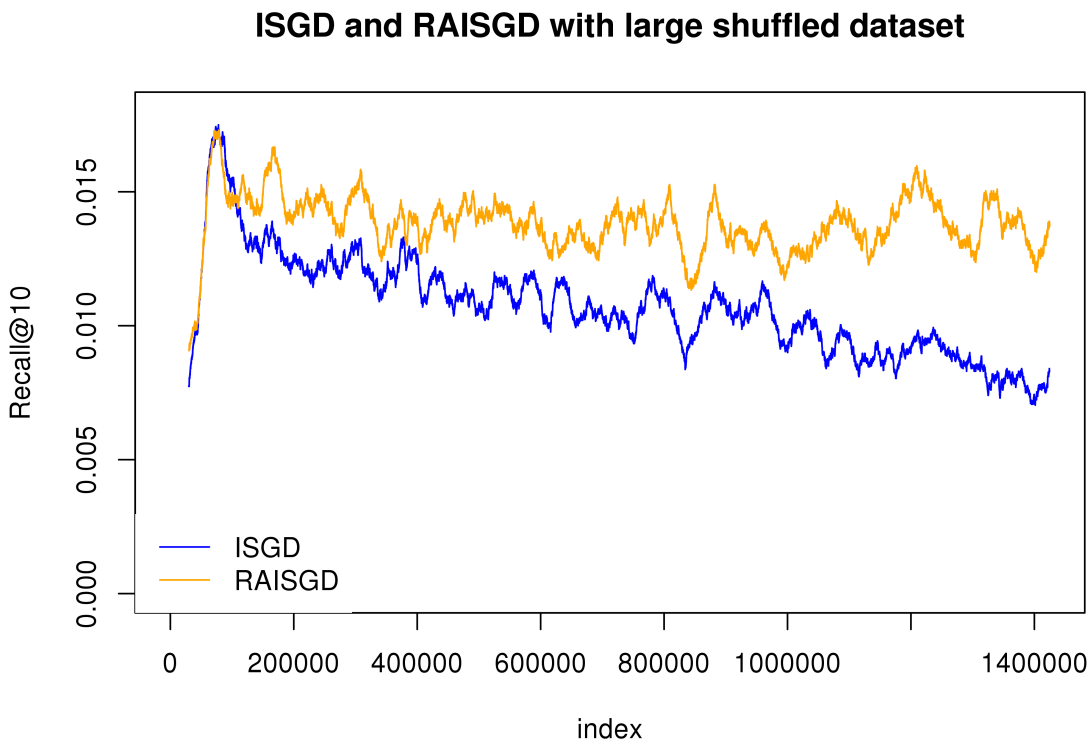


Figure 4.3: Evolution of Recal@10 of ISGD and RAISGD through the incremental learning and prediction process, illustrating the degradation of ISGD over time.

Our experiments in Section 5.7 show that RAISGD has a remarkably positive effect on the performance of the algorithm, with very low time overhead on the update operations, even with used with a high amount of negative feedback.

4.3.2 Rate-based algorithm

The above strategy uses the recency of occurrence of items, however it does not account for their frequency. This means that an item i that has occurred longer ago than an item j is more likely to be selected for negative feedback even if it has occurred many more times. To account for both recency and frequency we need a more elaborate scheme.

If we take the occurrences of each item as a non-stationary Poisson process, in our context, this is a counting process in which the interval between occurrences of a specific item is

distributed according to a Poisson distribution, with a non-stationary arrival rate z [1]. We model the occurrences of each item with a separate distribution and use an exponential decay function to gradually decrease the rate z of all items. Formally,

$$z_i^t = \alpha I_i^t + (1-\alpha) z_i^{t-1} \tag{4.7}$$

In (4.7), z_i^t is the arrival rate of item i at epoch t, I_i^t is an indicator function that returns 1 if i occurs at epoch t and 0 otherwise, and $0 < \alpha < 1$ is a decay factor. The decay is faster as α approximates 1.

This technique is used in quality control to obtain statistical control charts used to verify the normality of a process. A well known control chart for Poisson processes [Borror et al., 1998] is known as the Exponentially Weighted Moving Average Chart – or EWMA chart – uses (4.7). By monitoring the mean z, it is possible to detect when the process falls out of normal behavior. We do not use the concepts of statistical quality control, since we are only interested in having a simple way to monitor the rate of arrival of items. By maintaining the rate z for items, we can sort them by rate. Items with low rates are obviously less active and are better candidates to be used in negative examples. Note that the rates z are maintained globally, and do not depend on the user.

We use this model in ISGD, which results in Algorithm 4.5. On the arrival of a new user-item pair (u, i), we select the set of items $\{j_1, \ldots, j_l\}$ with the lowest arrival rates at the arrival time. This can be efficiently implemented using a simple priority queue. We then update the model with the positive example (u, i) and the imputed negative examples $\{(u, j_1), \ldots, (u, j_l)\}$. When updating the rates z We also mark the items used in negative examples as if they occurred naturally in the dataset. This increases their rates, avoiding the repeated penalization of items.

In (4.7) the parameter both the parameter α and the actual rate of occurrence of an item influence z, which is used as the priority of items. In practice, Algorithm 4.5 uses the two criteria to prioritize items: the recency and the multiplicity of occurrence. Items are less likely to be taken as negative examples as they have occurred recently and/or multiple times.

In our experiments in Section 5, we show that the parameter α has very little impact on the algorithm. This indicates that the algorithm does not benefit considerably from discriminating between items that have occurred very long ago. This means that the algorithm takes advantage from the recency criterion alone but very little benefit comes from maintaining actual rate of the item. Results in Section 5.7.2 confirm this and show that

[1] We use z instead of the conventional λ for the Poisson rate, to avoid confusion with the regularization factors used in several algorithms

Algorithm 4.5: RAISGD-RB: Recency-Adjusted ISGD (Rate-Based)

Data: a finite set or a data stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : k the no. of latent features
input : $iter$ the no. of iterations
input : λ the regularization factor
input : η the learn rate
input : l the number of negative examples
input : α the rate decay factor
output: A the user factor matrix
output: B the item factor matrix

$Q \leftarrow \texttt{initqueue}()$
for $(u,i) \in D$ **do**
  $I_i \leftarrow 1$
  if $u \notin \texttt{Rows}(A)$ **then**
    $A_u \leftarrow \texttt{Vector}(\text{size} : k)$
    $A_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin \texttt{Rows}(B)$ **then**
    $B_i \leftarrow \texttt{Vector}(\text{size} : k)$
    $B_i \sim \mathcal{N}(0, 0.1)$
  for $k \leftarrow 1$ **to** $\texttt{min}(l, \#\texttt{Rows}(B))$ **do**
    $j \leftarrow \texttt{dequeue}(Q)$
    for $count \leftarrow 1$ **to** $iter$ **do**
      $err_{uj} \leftarrow 0 - A_u \cdot B_j$
      $A_u \leftarrow A_u + \eta(err_{uj} B_j - \lambda A_u)$
    $I_j \leftarrow 1$
  for $k \leftarrow 1$ **to** $iter$ **do**
    $err_{ui} \leftarrow 1 - A_u \cdot B_i$
    $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$
  for $k \leftarrow 1$ **to** $\#\texttt{Rows}(B)$ **do**
    if $I_k = 1$ **then**
      $z_k \leftarrow \alpha + (1 - \alpha) z_k$
      $\texttt{enqueue}(Q, k)$
      $I_k \leftarrow 0$
    else
      $z_k \leftarrow (1 - alpha) z_k$

in terms of running time, there is a considerable time overhead when using Algorithm 4.5, when compared to Algorithm 4.4.

4.4 Summary

In this chapter we have proposed an incremental matrix factorization algorithm – ISGD (Algorithm 4.2) – to overcome the challenges of processing continuous flows of user feedback data, potentially overcoming many practical shortcomings of batch approaches to online recommendation problems. This contribution is important regarding its applicability in streaming environments, however it has a fundamental limitation. This limitation is related to the positive-only nature of the data. Without negative examples – information about items that users do not like –, a number of problems arise. For example, the trivial solution of $\hat{R}_{ui} = 1$ for all u, i would always yield minimal error, and no learning would occur. Moreover, it is extremely challenging to distinguish between bad recommendations – items that users do not like – and good recommendation candidates – items that users do not know. This is simply because the available data about the two is exactly the same: none.

Positive-only data only contains positive user-item interactions. The absence of a user-item pair can have two different meanings: either the user does not like the item, or the user does not know the item. Recommendation algorithms that learn from positive-only data need to be able to distinguish the two cases. In the first case, the item should not be recommended, while in the second case, the item is a candidate for recommendation. The problem is that the data does not contain negative feedback – items not liked by users. This problem is known as One-class Collaborative Filtering, given its resemblance with the problem of One-class Classification. One way to circumvent the problem is to address it as a learn-to-rank task, in which the task is to rank items according to their relevance to each user. This technique is described in Section 3.3.2.1 of Chapter 3. Another way, proposed in this chapter, is to artificially introduce negative feedback according to some criterion. Sampling and weighting techniques have been proposed for batch algorithms, but are not applicable for streaming data. Our proposal is to use a recency-based scheme, in which the least frequent items are chosen as negative feedback candidates. This technique fits seamlessly in our incremental framework for streaming user feedback, with potential to significantly avoid accuracy degradation. We have presented two algorithms, RAISGD (Algorithm 4.4) and RAISGD-RB (Algorithm 4.5), both able to effectively overcome the problems of dealing with positive-only feedback streams.

Chapter 5

Evaluation

One important contribution in this thesis is the evaluation methodology for recommender systems for streaming data. In our framework, classic evaluation protocols typically used to evaluate recommender systems are simply not applicable, given that we approach user feedback as a data stream. Fortunately, the evaluation of algorithms that learn from data streams is a well studied problem in the field of data stream mining [Gama, 2010; Gama et al., 2009, 2013; Bifet et al., 2015]. Our contribution in this context is a methodology that applies evaluation methods for data streams to the specific case of recommender systems. We show that a simple shift in the problem formulation – from the batch setting to a streaming one – has important implications in the evaluation methodology. We also illustrate how evaluating recommender systems in a streaming environment is useful in both the offline laboratory and real-world online scenarios.

We use the methodology described below in Section 5.1 to assess the accuracy and runtime performance of the algorithms presented in Chapters 3 and 4. Section 5.4 describes the details of our experimental process, datasets, hyperparameter estimation. The actual results are divided in three sets of experiments. First, in Section 5.5 we measure the benefits of using the incremental ISGD (Algorithm 4.2), by comparison with a batch version of the same algorithm (Algorithm 2.1). Second, in Section 5.7, we assess the effectiveness of the strategy presented in Chapter 4 by comparing RAISGD (Algorithm 4.4) with ISGD. Finally, in Section 5.8, we provide a thorough comparison between ISGD, RAISGD, a classic user-based neighborhood algorithm (Algorithm 3.1 and 3.2 - UKNN) and two versions – BPRMF and WBPRMF – of a state-of-the-art learning-to-rank factorization alternative (Algorithm 3.4). We evaluate two dimensions of the algorithms' performance: accuracy and time. Although our main objective is to maximize accuracy, this cannot be done at any expense of time. It is extremely important that algorithms are able to timely process data streams, otherwise they are simply not useful in practice. We have published the core of the methodology presented in this Chapter in [Vinagre et al., 2014a].

5.1 Offline evaluation methodologies

Offline evaluation usually refers to evaluation of algorithms done with archived data, without the interaction of users. Such evaluation methodologies are in opposition to *online evaluation* [Kohavi et al., 2009], which is done in real time with real users.

Offline protocols allow researchers to evaluate and compare algorithms by simulating user behavior. This typically begins by splitting the ratings dataset in two subsets – training set and testing set – randomly choosing data elements from the initial dataset. The training set is initially fed to the recommender algorithm to build a predictive model. To evaluate the accuracy of the model, different protocols can be used. Generally, these protocols group the test set by user – or user session – and "hide" user-item interactions randomly chosen from each group. These hidden interactions form the hidden set. Rating prediction algorithms are usually evaluated by comparing predicted ratings with the hidden ratings. Item recommendation algorithms are evaluated performing user-by-user – or session-by-session – comparison of the recommended items with the hidden set.

The most common protocols using this strategy are *All-but-N* and *Given-N* [Breese et al., 1998]. The All-but-N protocol hides exactly N items from each user in the test set. One popular sub-protocol is the *All-but-One* protocol, which hides exactly one item from each user in the test set. The *Given-N* protocol keeps exactly N items in the test set and hides all others. In both protocols, hidden ratings are randomly chosen from each user.

Offline protocols try to simulate user activity. They are usually easy to implement, and enable the reproducibility of experiments, which is a key feature for peer-reviewed research. However there are a few limitations to consider:

- *Dataset ordering*: randomly selecting data for training and test, as well as random hidden set selection, shuffles the natural sequence of the data. Algorithms designed to deal with naturally ordered data cannot be rigorously evaluated if datasets are shuffled. One straightforward solution is simply not to shuffle data. That is, to pick a moment in time or a number of ratings in the dataset as the split point. All ratings given before the split point are used to train the model and all subsequent ratings are used as testing data. One awkwardness with this approach is how to select the hidden set. In [Shani and Gunawardana, 2011] and [Lathia, 2010] the authors suggest that all ratings in the test set should be hidden;
- *Time awareness*: shuffling data potentially breaks the logic of time-aware algorithms. For example, by using future ratings to predict past ratings. This issue may as well be solved by keeping the chronological order of data;
- *Incremental updates*: incremental algorithms perform incremental updates of their

models as new data points become available. This means that neither models or training and test data are static. Models are continuously being readjusted with new data. As far as we know to this date, the only contributions in the field of recommender systems that explicitly address this issue are [Vinagre et al., 2014b] and [Siddiqui et al., 2014]. This issue has already been addressed in the field of data stream mining [Gama et al., 2009, 2013];

- *Session grouping*: most natural datasets, given their unpredictable ordering, require some pre-processing to group ratings either by user or user session in order to use offline protocols. As data points accumulate, it eventually may become too expensive to re-group them. This is true also for any other kind of data pre-processing task;

- *Recommendation bias*: in online production systems, user behavior is – at least expectedly – influenced by recommendations themselves. It is reasonable to assume, for instance, that recommended items will be more likely followed than if they were not recommended. Simulating this offline usually requires complicated user behavior modeling which can be expensive and prone to systematic error. One way to evaluate the actual impact of a recommender system is to conduct user surveys and/or A/B testing [Kohavi et al., 2009; Domingues et al., 2013; Knijnenburg et al., 2012; Pu et al., 2012].

The above limitations, along with other known issues [Shani and Gunawardana, 2011; McNee et al., 2006; Herlocker et al., 2004], weaken the assumption that user behavior can be accurately modeled or reproduced in offline experiments. From a business logic perspective [Félix et al., 2014] offline evaluation may also not be timely enough to support decision making. These issues motivate the research of alternative or complementary evaluation methodologies.

In [Shani and Gunawardana, 2011] some clues are provided on how to solve some of these problems. One straightforward solution to the first two problems is simply not to shuffle data – or if timestamps are available, pre-order ratings accordingly. Then a moment in time or a number of ratings is chosen as the split point for the dataset. All ratings given before the split point are used to train the model and all subsequent ratings are used as testing data. One problem with this approach is how to select the hidden set. In [Shani and Gunawardana, 2011] and [Lathia, 2010] the authors suggest that all ratings in the test set should be hidden, assuming that users already have had activity before the split point – those who had not are simply ignored. Another possibility is to adapt the *Given-N* and *All-but-N* protocols to preserve order. The hidden set can contain the last N items for each user – *All-but-N* – or hiding all but the first N items – *Given-N*.

5.2 Prequential evaluation

To solve the issue of how to evaluate algorithms that continuously update models, we propose the usage of a prequential methodology [Gama et al., 2013]. Evaluation is made treating incoming user feedback data as a data stream. Evaluation is continuously performed in a test-then-learn scheme (Fig. 5.1): whenever a new rating arrives, the corresponding prediction is scored according to the actual rating. This new rating is then used to update the model.

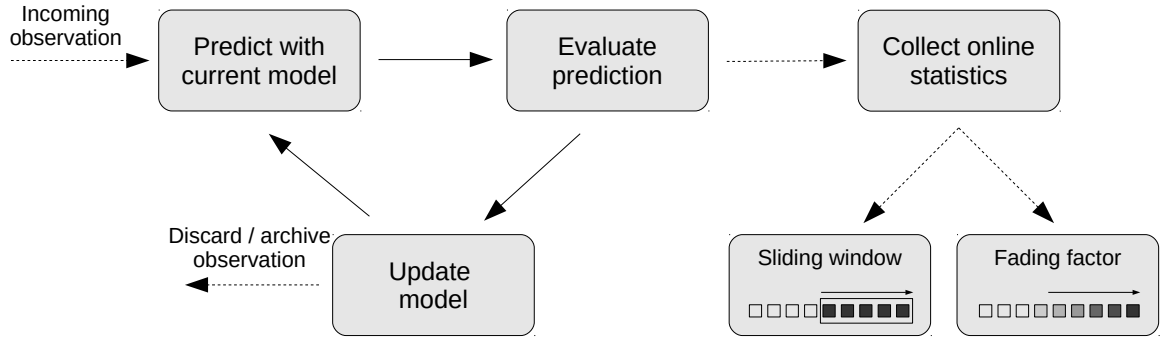


Figure 5.1: Prequential evaluation

In our particular setting, since we are using positive-only data, observations do not contain actual ratings. Instead, each observation consists of a simple user-item pair (u, i) that indicates a positive interaction between user u and item i. The following steps are performed in the prequential evaluation process:

1. If u is a known user, use the current model to recommend a list of items to u, otherwise go to step 3;
2. Score the recommendation list given the observed item i;
3. Update the model with (u, i) (optionally);
4. Proceed to – or wait for – the next observation

One important note about this process is that it is entirely applicable to algorithms that learn either incrementally or in batch mode. This is the reason why step 3. is annotated as optional. For example, instead of performing this step, the system can store the data to perform batch retraining periodically.

This protocol provides several benefits over traditional batch evaluation:

- It allows continuous monitoring of the system's performance over time;

- Several metrics can be captured simultaneously;
- If available, other kinds of user feedback can be included in the loop;
- Real-time statistics can be integrated in the algorithms' logic – e.g. automatic parameter adjustment, drift/shift detection, triggering batch retraining;
- In ensembles, relative weights of individual algorithms can be adjusted;
- The protocol is applicable to both positive-only and ratings data;
- By being applicable both online and offline, experiments are trivially reproducible if the same data sequence is available.

In an offline experimental setting, an overall average of individual scores can be computed at the end – because lab datasets are inevitably finite – and on different time horizons. For a recommender running in a production system, this process allows us to follow the evolution of the recommender by keeping online statistics of any number of chosen metrics. Thereby it is possible to depict how the algorithm's performance evolves over time. In this chapter, wherever applicable, we present both the overall average score and complement it with plots of the evolving score using a simple moving average of the metric(s).

5.2.1 Limitations

A challenging aspect of this method is that it only evaluates over a single item at each step, potentially failing to recognize other possible good recommendations. If item i is not recommended at the time the observation is made, the score will naturally be 0. However, other items within the N recommendations may occur in future observations for that user. In other words, the protocol exclusively evaluates how well the model predicts the next observation, ignoring all subsequent ones. Although this is a somewhat challengingly strict protocol, we have performed experiments by matching the recommended items with not just the current, but all future observations for each user – only possible offline –, and found that overall scores do not improve substantially. However, this strictness of the protocol may potentially have a higher impact with other metrics or data. One way to relax this, is to match the active observation not only with the current prediction, but also with a set of previous predictions. One other possible approach is to use a hybrid evaluation method such as in [Siddiqui et al., 2014].

5.3 Online evaluation

Offline evaluation is important to assess the predictive ability and runtime performance of algorithms. However it is arguable that it is enough[McNee et al., 2006]. There is no guarantee that an algorithm with good offline results will have good online performance, from the users' perspective. The only way to perform user-centric evaluation is to interact with real users [Pu et al., 2012]. The main disadvantage of online evaluation is that it is generally not reproducible, because it requires the interaction with a usually large number of users whose behavior is naturally unpredictable.

Prequential evaluation is essentially designed to run online, with live data streams. One of the benefits is precisely that it can be used to perform online measurements on the user side for posterior analysis. Additionally these measurements are available in real time, which can be useful, for instance to perform automatic online adjustments, or to give stakeholders the ability to make informed decisions.

In this thesis, we do not perform online evaluation. In order to be valuable, online evaluation requires access to online production systems with a large number of users, which is not widely available.

5.4 Experimental process details

In this section we provide the details on our experimental process, using the prequential approach described in Section 5.2. We describe the datasets, metrics and details of the methodology.

5.4.1 Datasets

To simulate a streaming environment we need datasets that maintain the natural order of the data points, as they were generated. Additionally, we need positive-only data, since our algorithms are not designed to deal with ratings. Nowadays, a considerable number of datasets is publicly available for research in recommender systems, but few conciliate these two requirements – positive-only and naturally ordered. We were able to collect a total of 6 datasets, described in Table 5.1, that we use in several experiments. ML1M and ML10M are based on the well known Movielens-1M and Movielens-10M movie rating datasets [Grouplens, 2013]. LFM-50 is a subset consisting of a random sample of the activity of 50 users taken from the Last.fm[1] dataset, originally collected and used by Celma

[1] http://last.fm/

Dataset	Domain	Application	Events	Repeated	Users	Items	Sparsity
PLC-PL	Music	Playlisting	111 942	no	10 392	26 117	99.96%
PLC-STR	Music	Streaming	588 851	yes	7 580	30 092	99.74%
LFM-50U	Music	Streaming	1 121 520	yes	50	159 208	85.91%
YHM-6KU	Music	Rating	476 886	no	6 000	127 448	99.94%
ML1M	Movies	Rating	226 310	no	6 014	3 232	98.84%
ML10M	Movies	Rating	1 544 812	no	67 312	8 721	99.74%

Table 5.1: Dataset description

[Celma, 2010, 2013]. To obtain the YHM-6KU, we sample the activity of 6000 users from the Yahoo! Music dataset [Yahoo, 2013], originally collected for the KDD Cup 2011 [Dror et al., 2012]. The two remaining datasets PLC-PL and PLC-STR are activity logs of two types taken from Palco Principal[2], a portuguese social network for non-mainstream artists and fans. PLC-PL is the playlisting dataset, in which each user-item pair consists of a user adding a music track to a personal playlist, and PLC-STR is the music streaming log, in which a user-item pair is generated every time a user listens to a music track.

All of the 6 datasets consist of a chronologically ordered sequence of positive user-item interactions. One important note about ML1M, ML10M and YHM-50U is that their original versions are ratings datasets. In order to use them as positive-only data, we retain the user-item pairs for which the rating is in the top 20% of the rating scale of each dataset. This means that only the user-item pairs with the rating 5 – from a 1 to 5 rating scale – in Movielens-1M and Movielens-10M and 80 or more in the YHM-6KU dataset – in a scale from 0 to 100 – are retained. Naturally, only single accurrences of user-item pairs are available in these datasets, since users do not rate the same item more than once. PLC-PL shares this characteristic, because it results of users adding music tracks to personal playlists, which is only recorded once in the available dataset.

All other datasets have multiple occurrences of the same user-item pairs. We interpret these interactions a positive opinion of a user about an item. Note that in some cases, this assumption may be arguable. For example, take the case of a music streaming dataset (three of our datasets consist of music listening events of this type). The fact that a user listened to a particular music track does not necessarily mean that the user likes this track. The same could happen in an e-commerce application page view log. The user may be exploring the catalog with no particular interest in the items she visits. In [Vinagre et al., 2014b], [Vinagre et al., 2015a] and [Vinagre et al., 2014a] we eliminate single occurrences of user-item pairs in music streaming datasets, since we assume that a user listening to a music track exactly once is a very weak indication a positive interaction – and may actually indicate a negative opinion. In this thesis, we do not do that for two reasons. First, this is an artificial data pre-processing step not trivially reproducible in a real-world streaming

[2] http://www.palcoprincipal.com/

environment. Second, the true positive interactions are naturally much more frequent, which means that the algorithms learn from them repeatedly, diluting the effect of non-representative examples.

5.4.2 Experimental process

Using the datasets above, we run a set of experiments using the prequential approach described in Section 5.2. To kickstart the evaluation process with incremental algorithms, we have two options:

(a) begin with zero knowledge, using 100% of the data to perform incremental training and to evaluate algorithms;

(b) use part of the available data to perform an initial batch training of the algorithm, and perform incremental training and evaluation on the remaining data.

We choose option **(b)** for two reasons. First, although our evaluation is performed offline, we want to simulate a scenario as close as possible to an online setting. It is unlikely that an online system does not have already data available, especially given that we are dealing with positive-only feedback. Second, we want to avoid *cold-start* problems, that may affect algorithms and datasets differently. This way, all algorithms begin incremental learning with a fully functional model. Since we are mostly interested in the evaluation of the incremental process, we use a small part of the data to do batch training. In all experiments, we use the first 10% observations to train the algorithms in batch. This way, we are still able to evaluate on 90% of each dataset.

In our recommendation setting, we assume that items that users have already co-occurred with – i.e. items that users know – are not good candidates for recommendation. In some applications this may not be desirable – e.g. music playlist recommendation, restaurants. However, we believe that the most common recommendation task is to recommend unknown items to users, rather than items that they already know. This has one important implication in the prequential evaluation process, on datasets that have multiple occurrences of the same user-item pair. Evaluation at these points is necessarily penalized, since the observed item will be not be within the recommendations. In such cases, we bypass the scoring step, but still use the observation to update the model.

5.4.3 Metrics

We measure two dimensions on the evaluation process: accuracy and time. To use the prequential process described in Section 5.2, we need to make a prediction and evaluate it

at every user-item pair (u, i) that arrives in the data stream. To do this, we use the current model to recommendation N items to user u. We then score this recommendation list, by matching it to the actually observed item i. We use a recommendation list with length $N = 20$, and then score this list – or a sublist – as 1 if i is within the recommended items, and 0 otherwise. Using the whole list, this is equivalent to Recall with a cutoff of 20 – or Recall@20. However we can also cutoff the bottom of the list and obtain Recall@C with any C between 1 and 20. In our experiments we use Recall@C with $C \in \{1, 5, 10, 20\}$. Because only one item is tested against the list, Recall@C can only take the values $\{0, 1\}$. We can calculate the overall Recall@C by averaging the scores at every step. Additionally, we depict it using a moving average of Recall@10 in a series of plots. Time is measured mainly in milliseconds at every step and we depict it using the same techniques we use with accuracy.

5.4.4 Statistical significance

Several illustrative examples of the application of statistical significance tests in algorithms that learn from data streams have been presented in [Gama et al., 2013]. To assess the significance of the results, we use the signed McNemar test over a sliding window. The McNemar test assesses the statistical significance of the difference between results obtained by two algorithms that learn from the same data, and is especially suited for our measurement – at each point Recall is either 0 or 1, which resembles a 0-1 loss function. We need to maintain two quantities relative to the outcome of two algorithms A and B: $n_{0,1}$, which denotes the number of examples in which Recall is 0 for algorithm A and 1 for Algorithm B, and $n_{1,0}$ which denotes the number of examples for which Recall is 1 for algorithm A and 0 for algorithm B. These quantities can be easily maintained online, which is a requirement for streaming data. The test can be performed at every step of the prequential process by calculating the statistic:

$$M = sign(n_{0,1} - n_{1,0}) \times \frac{(n_{0,1} - n_{1,0})^2}{n_{0,1} + n_{1,0}} \tag{5.1}$$

M follows a χ^2 distribution with 1 degree of freedom. The null hypothesis – e.g. algorithms A and B are not significantly different – is rejected if $|M| > 6.635$, for a confidence level of 99%. Having rejected the null hypothesis, the sign of M tells us which of both algorithm performs better.

5.4.5 Parameter optimization

All algorithms used in the experiments have a number of user-defined parameters – or hyperparameters. Typically, the optimal values of the parameters are data dependent and are obtained by performing cross-validation on the training set. As mentioned above, cross-validation is not applicable to data streams. This is because we do not assume that users generate data with stationary distributions. Instead, the underlying distributions in a data stream are subject to changes over time – concept drifts. As a consequence, the optimal values for the hyperparameters are also likely to change over time. In a real-world system, optimal hyperparameter settings would need to be continuously assessed, using concept drift detection techniques [Gama et al., 2004].

Drift detection techniques are not used in this thesis. Instead, we obtain optimal hyperparameter setting on the initial 10% of the data that is used for the initial batch training of the algorithms, and use the same values for the entire experiment. We use the exact same process used in evaluation described above: out of the first 10% of all the available data, we use the first 10% for batch training and the remaining 90% for prequential evaluation. We perform a grid search on the hyperparameters that maximize accuracy for each algorithm with each one of the datasets.

5.4.6 Presentation of results

We present results obtained in our experiments using a table with overall averages for Recall@C with $C \in \{1, 5, 10, 20\}$ and average processing time in milliseconds obtained by each algorithm and dataset. The tables that summarize the results are informative about the overall performance of the algorithms, however they do not tell us much about the dynamics of the process. For instance, two algorithms may yield the same average Recall for a given dataset, but may have opposite trends – i.e. one algorithm may tend to improve over time, while other may tend to degrade. Averages hide these dynamics. Prequential evaluation allows us to maintain statistics at every step of the prequential process. In our experiments, we take advantage of this by plotting a moving average of Recall, that depicts how accuracy evolves over time, enabling the analysis of the algorithms' learning process, that would otherwise remain unnoticed. Unless noted otherwise, we use moving averages with $n = 10000$ that depict the evolution of the ongoing learning process of each algorithm using Recall@10. This type of graphical presentation type is more informative, in the sense that it is able to compare not only the overall relative performance of algorithms, but also the dynamics of each algorithm's performance over time.

Statistical significance tests are also conducted with respect to the dynamic aspect of the learning process with streaming data. Where applicable, we illustrate the pairwise

comparison between algorithms using the signed McNemar test described in Section 5.4.4, by continuously calculating M over a sliding window with 10 000 examples, the same length of the moving average used to plot the evolving accuracy.

5.4.7 Software and hardware

All algorithms evaluated in this thesis are implemented in the MyMediaLite software library [Gantner et al., 2011], originally available from `https://github.com/zenogantner/MyMediaLite`. The exact code used in this thesis is available from `https://github.com/joaoms/MyMediaLite/tree/v3.04-jvinagre-thesis`. The underlying operating system is CentOS Linux release 7.1.1503 with the .NET implementation package Mono version 4.0.4 (Stable 4.0.4.1/5ab4c0d). The experiments were run in a homogeneous set of machines with the exact same hardware and software configuration. All machines are configured with two Intel Haswell CPU cores running at 2.30GHz, and 12GB RAM. To ensure the reliability of processing time measurements, each experiment was given exclusive access to exactly one core, and all possible concurrent processes were disabled.

5.5 Comparing incremental and batch learning

In Chapter 4 we present ISGD (Algorithm 4.2). This is basically an incremental version of Algorithm 2.1 (BSGD) that works with positive-only feedback. In this section we compare ISGD against BSGD. However, a fair comparison is not a trivial evaluation task. Our hypothesis is that incremental algorithms are beneficial in terms of accuracy and processing time.

To perform a fair comparison, we assume the following:

- the algorithms have access to the same data at the same time;
- batch and/or incremental training are possible and optional at any time;
- the same recommendation requests are made to both algorithms, and in the same sequence.

Using prequential evaluation, we can use the first 10% of each dataset to perform an initial batch training of the two algorithms. This means that the algorithms begin with the same initial model. Then, ISGD learns incrementally from each observed data point. However BSGD is unable to do that, which is an obvious disadvantage. However, given our first assumption, we ensure that the same data is available for both algorithms at all times, and

that model updates are allowed at any time. This means that BSGD has the possibiity to perform batch retraining of the model with all the available data at any point. The most extreme scenario is to perform batch training of the model for every user-item pair that arrives in the stream. Obviously, this is an increasingly heavy operation – unless some sampling strategy is used – and is not a realistic scenario.

To maintain objectivity, we ask the following question:

> How frequently does BSGD need to be retrained to avoid accuracy degradation?

By answering this question, we are able to compare ISGD and BSGD in terms of how expensive it is to maintain acceptable performance.

5.5.1 Results

In Table 5.2 we present average Recall@N, obtained by BSGD – trained at different intervals – and ISGD. The experiment is performed by using the ISGD with incremental updates for every example in the data stream. We retrain BSGD in batch, using all observed data points, every 100, 1000, 10000 examples for all datasets, and additionally at intervals of 5 and 10 for the shortest datasets – PLC-PL and ML1M. We do not test for shorter intervals in the 4 largest datasets because it is very time consuming, and it adds very little information. The time column in the table indicates the training time per example, i.e. the total time spent by the algorithm retraining the model, divided by the total number of examples in the dataset. Obviously, BSGD requires more and more time per example as the batch retraining interval becomes shorter and shorter. In all cases, including the datasets with the shortest training intervals of 5 – batch retraining every 5 observations –, Recall remains considerably below the one obtained by ISGD.

We also present Figure 5.2 that shows how the evolving Recall@10 of BSGD and ISGD in the same experiments, using a moving average with $n = 10000$. This figure confirms results in Table 5.2, however it is possible to observe how Recall@10 varies over time.

5.5.2 Discussion

By analyzing Table 5.2 and Figure 5.2 it becomes clear that to obtain accuracy comparable to ISGD, BSGD requires time several orders of magnitude above the time required by ISGD. Even in the two datasets where we are able to measure the accuracy of BSGD by retraining it every 5 examples – which is already not practical in a realistic scenario – its accuracy remains considerably below ISGD. Looking closely to the time requirements of BSGD as

Dataset	Algorithm	Recall@1	Recall@5	Recall@10	Recall@20	Time
PLC-PL	BSGD 10000	0.004	0.011	0.015	0.023	0.164
	BSGD 1000	0.007	0.019	0.027	0.038	1.445
	BSGD 100	0.011	0.031	0.045	0.061	14.744
	BSGD 10	0.031	0.080	0.110	0.144	141.869
	BSGD 5	0.040	0.105	0.141	0.181	298.160
	ISGD	0.104	0.199	0.234	0.265	0.535
PLC-STR	BSGD 10000	0.002	0.006	0.010	0.016	1.403
	BSGD 1000	0.003	0.010	0.015	0.024	13.792
	BSGD 100	0.008	0.025	0.036	0.053	137.679
	ISGD	0.127	0.241	0.277	0.302	0.237
LFM-50U	BSGD 10000	$<$0.001	$<$0.001	$<$0.001	0.001	1.132
	BSGD 1000	$<$0.001	$<$0.001	0.001	0.002	11.477
	BSGD 100	0.001	0.003	0.004	0.006	112.974
	ISGD	0.034	0.049	0.052	0.055	2.625
YHM-6KU	BSGD 10000	$<$0.001	0.001	0.002	0.003	1.867
	BSGD 1000	$<$0.001	0.001	0.003	0.005	18.779
	BSGD 100	0.001	0.002	0.004	0.007	186.393
	ISGD	0.030	0.063	0.082	0.103	4.462
ML1M	BSGD 10000	$<$0.001	0.002	0.003	0.005	0.560
	BSGD 1000	0.001	0.003	0.005	0.009	5.245
	BSGD 100	0.001	0.005	0.009	0.017	52.342
	BSGD 10	0.002	0.008	0.014	0.026	522.549
	BSGD 5	0.002	0.008	0.015	0.028	$>$1000
	ISGD	0.005	0.021	0.034	0.055	0.069
ML10M	BSGD 10000	$<$0.001	0.002	0.003	0.006	2.066
	BSGD 1000	$<$0.001	0.001	0.002	0.004	20.959
	BSGD 100	0.001	0.003	0.006	0.011	223.792
	ISGD	0.007	0.026	0.040	0.061	0.176

Table 5.2: Comparison between BSGD and ISGD. BSGD is retrained at fixed intervals of 100, 1000 and 10 000 (additionally 5 and 10 for PLC-PL and ML1M) data points in the stream. ISGD is retrained incrementally with every new example. Time per point is measured in milliseconds as the total time spent in training divided by the number of examples in each dataset.

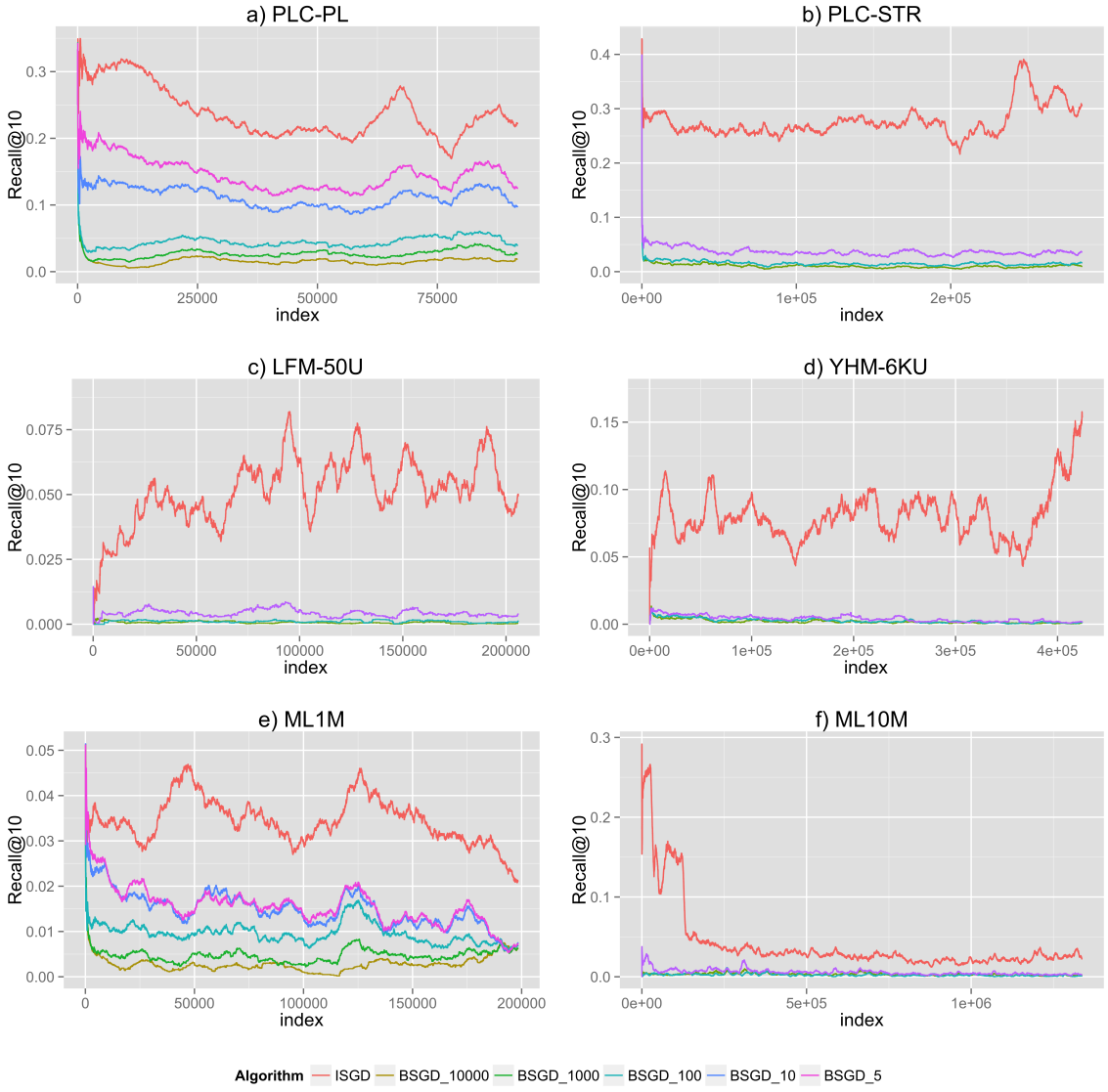


Figure 5.2: Prequential evaluation of Recall@10 with BSGD and ISGD. BSGD is retrained at fixed intervals of 100, 1000 and 10 000 (additionally 5 and 10 for PLC-PL and ML1M) data points in the stream. ISGD is retrained incrementally with every new example. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

different intervals, we see it behaves quite predictably. Time roughly grows according to the number of times the algorithm is retrained. Based on the observations, it is safe to state that to obtain level of accuracy similar to ISGD, BSGD requires too much time to be applicable for streaming data.

5.6 ISGD with bagging

In Section 4.1.1, we introduce incremental bagging as a strategy to improve the accuracy of ISGD. In this section, we evaluate the impact of bagging in terms of accuracy and time. To evaluate bagging, we experiment with three levels of bootstrapping – the number of bootstrap submodels – $m \in \{8, 16, 32, 64\}$. Table 5.3 summarizes the results of our experiments[3]. With all datasets except YHM-6KU, bagging improves the Recall, especially with $m \geq 32$. One interesting observation is that bagging has a bigger influence on higher Recall cutoffs, which suggests that improvements of the predictive ability are typically not obtained in the top 5 recommended items.

The model update times increase approximately proportionally to the number of bootstrap nodes m, which is not surprising, since the algorithm performs the update operations in average one time in each one of the m submodels. However, since the update time is small in all cases, this overhead may be manageable in many applications. The last column of Table 5.3 contains recommendation times, which is the time required to produce a recommendation list. This is important, because the bagging algorithm needs to average predictions coming from all m submodels, which obviously introduces an overhead. Results show that both the update times and recommendation times increase proportionally to m. However, the recommendation step is a far more costly operation, even considering that it may be computed with parallel code[4]. For example, using $m = 64$ with LFM-50U and YHM-6KU, recommendations are computed in nearly two seconds in average.

5.6.1 Discussion

Results in Table 5.3 and Figure 5.3 show that bagging clearly improves the accuracy of ISGD. This improvement is mainly observable with cutoffs $N > 1$ of Recall. This is especially visible when we compare Figure 5.3, that presents results with Recall@10, with Figures B.1, B.2 and B.3 – in Appendix B. Given that bagging is mainly used to reduce variance [Breiman, 1996], this suggests that the variance of ISGD is smaller in roughly the top 5 recommendations. Another observation, is that improvements are not consistent with all datasets. With LFM-50U, for example, bagging only slightly outperforms the baseline ISGD – and only with $m \geq 32$ –, while with ML1M, the improvement is much higher in proportion, even with lower m.

It is also clear that the time overheads are considerable, mainly at recommendation time,

[3] We do not present results with $m \geq 32$ for ML10M, given that this dataset is much larger than the remaining and we did not have enough computational resources for it at the time of the experiments

[4] Experiments with bagging were exceptionally run with parallel code in machines with 4 cores, with the same specifications of the hardware described in Section 5.4.7.

Dataset	m	Rec@1	Rec@5	Rec@10	Rec@20	Upd. (ms)	Rec. (ms)
PLC-PL	ISGD	**0.104**	0.198	0.234	0.265	0.535	9.537
	8	0.069	0.162	0.211	0.260	4.849	35.673
	16	0.081	0.187	0.240	0.294	7.289	43.782
	32	0.087	0.200	0.254	0.308	21.012	98.329
	64	**0.093**	**0.209**	**0.267**	**0.324**	38.673	226.730
PLC-STR	ISGD	**0.127**	**0.241**	0.277	0.302	0.237	21.736
	8	0.076	0.194	0.257	0.316	2.563	64.793
	16	0.081	0.215	0.284	0.349	4.732	132.812
	32	0.088	0.229	0.302	0.370	9.508	264.846
	64	0.092	0.237	**0.313**	**0.384**	18.012	517.479
LFM-50U	ISGD	**0.034**	0.049	0.052	0.055	2.625	94.177
	8	0.023	0.044	0.052	0.058	21.449	241.452
	16	0.026	0.050	0.059	0.066	43.094	491.689
	32	0.028	0.055	0.064	0.071	84.536	984.060
	64	0.030	**0.057**	**0.067**	**0.075**	168.781	1.958 s
YHM-6KU	ISGD	**0.030**	**0.063**	0.082	0.103	4.462	89.321
	8	0.011	0.033	0.051	0.076	28.529	347.422
	16	0.012	0.037	0.058	0.086	54.723	667.898
	32	0.019	0.055	0.082	0.117	158.744	990.551
	64	0.021	0.059	**0.087**	**0.123**	328.924	1.934 s
ML1M	ISGD	0.005	0.021	0.034	0.055	0.069	2.557
	8	0.005	0.019	0.033	0.056	0.517	7.208
	16	0.006	0.022	0.038	0.063	1.390	21.816
	32	0.006	0.025	0.042	0.071	1.866	33.496
	64	**0.007**	**0.026**	**0.045**	**0.074**	3.999	41.090
ML10M	ISGD	**0.007**	0.026	0.040	0.061	0.176	2.162
	8	0.006	0.023	0.039	0.063	2.605	18.160
	16	**0.007**	**0.027**	**0.046**	**0.073**	4.083	27.735

Table 5.3: Comparison between ISGD with and without bagging. m indicates the bootstrap level (the number of bootstrap nodes). The last two columns contain the average update times and the average recommendation times.

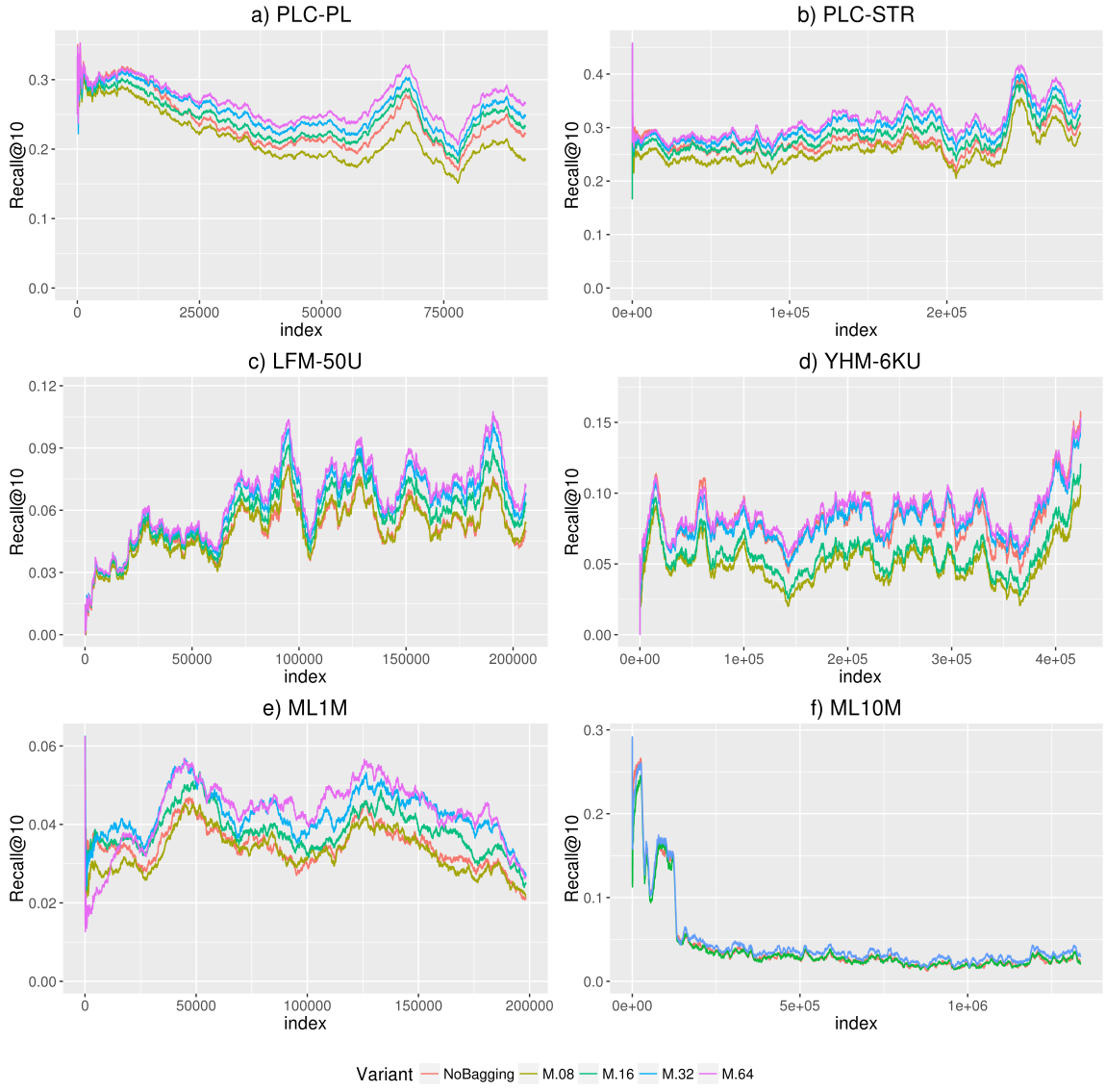


Figure 5.3: Prequential evaluation of Recall@10 with ISGD with and without bagging. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10000 points are drawn using the accumulated average.

when aggregating results from the m bootstrap models. Naturally, this overhead grows linearly with the number of bootstrap models. In many cases, this not be suitable for practical applications. The model update times suffer the same impact, but in practice this impact is not so relevant, given that update times remain low (less than 400ms). Fortunately, as most ensemble techniques, parallel processing can be trivially used to alleviate the time overhead. Additionally, there may be room for code optimization or approximation methods that require less and/or more efficient computations.

5.7 ISGD with recency-based negative feedback

In Chapter 4, we introduce the problem of using ISGD with positive-only feedback. ISGD is an incremental version of BSGD, an algorithm originally developed for ratings data. The problem arises from the fact that ISGD corrects the model to minimize error with the respect to the same value 1, always. The absence of negative examples leads to a model that eventually loses discriminative power, given that all predictions tend to accumulate very close to the same value 1. This naturally is a limitation of ISGD. To solve this problem, we propose RAISGD (Algorithm 4.4 in Chapter 4, that artificially introduces *negative* user-item pairs in the stream. For these pairs, the prediction of the model is corrected to approximate 0 instead of 1, as we do for the regular positive pairs.

To illustrate the phenomenon, we compare ISGD and RAISGD using all six datasets in Table 5.1. We tune the hyperparameters using the methodology described in Section 5.4.5. Table 5.6 contains the optimal settings for both algorithms. Because RAISGD is ISGD with the extra feature of using negative feedback, all parameters match, except l, which is the amount of negative feedback used in RAISGD. Notice that with ML1M, the optimal value for negative feedback amount is $l = 0$, which in practice means that RAISGD and ISGD are equivalent. To be able to observe the impact of negative feedback with ML1M, we use $l = 1$ even though it is not the optimal setting for that dataset.

Overall results are in Table 5.4. Improvements in Recall are visible in PLC-PL, PLC-STR, LFM-50U, YHM-6KU and ML10M. The exception is ML1M, with which negative feedback results in worse performance. We can also plot the evolution of accuracy of the algorithms over time. Figure 5.4 depicts the moving average of Recall@10 with all datasets using different values for l. Using this visual presentation, results in Table 5.4 can generally be confirmed, however it becomes more obvious that differences are not always big and in some regions the alternatives are almost indistinguishable. To illustrate the significance of the differences we show in Figure 5.5 the results of the McNemar test with respect to the baseline ISGD algorithm – i.e. RAISGD with $l = 0$ –, for the exact same experiments depicted in 5.4.

Figures similar to 5.4 and 5.5 for Recall@$\{$1,5,20$\}$ are available in Annex B.

5.7.1 Impact in processing time

In terms of processing time, RAISGD is naturally more demanding, since it needs to process l more data points than ISGD. However, looking at the average update times in Table 5.4 the overhead does not seem to be very problematic. In all cases, even with $l = 10$, that forces the algorithm to process 10 additional data points for each (u, i) pair in the data stream, the

Dataset	l	Recall@1	Recall@5	Recall@10	Recall@20	Time
PLC-PL	0 (ISGD)	0.104	0.199	0.234	0.265	**0.535**
	1	**0.115**	**0.211**	**0.249**	**0.282**	0.586
	2	0.088	0.165	0.199	0.228	0.714
	3	0.050	0.108	0.135	0.157	0.754
	5	0.027	0.068	0.090	0.109	0.809
	10	0.026	0.065	0.086	0.105	0.949
PLC-STR	0 (ISGD)	0.127	0.241	0.277	0.302	**0.237**
	1	**0.172**	**0.285**	**0.322**	**0.353**	0.273
	2	0.159	0.253	0.285	0.310	0.379
	3	0.103	0.157	0.178	0.197	0.414
	5	0.021	0.037	0.044	0.051	0.483
	10	0.010	0.021	0.026	0.030	0.655
LFM-50U	0 (ISGD)	0.034	0.049	0.052	0.055	2.625
	1	**0.044**	**0.058**	**0.062**	**0.065**	**2.333**
	2	0.036	0.047	0.050	0.053	3.008
	3	0.022	0.030	0.033	0.036	3.017
	5	0.007	0.012	0.014	0.015	3.069
	10	0.006	0.010	0.012	0.013	3.185
YHM-6KU	0 (ISGD)	0.030	0.063	0.082	0.103	**4.462**
	1	**0.037**	**0.070**	**0.090**	**0.112**	4.591
	2	0.028	0.060	0.079	0.103	5.181
	3	0.016	0.042	0.059	0.075	5.257
	5	0.003	0.006	0.008	0.010	5.376
	10	0.001	0.003	0.004	0.006	5.707
ML1M	0 (ISGD)	**0.005**	**0.021**	**0.034**	**0.055**	**0.069**
	1	0.003	0.013	0.022	0.039	0.101
	2	0.002	0.010	0.019	0.035	0.203
	3	0.002	0.009	0.016	0.031	0.248
	5	$<$0.001	0.001	0.001	0.002	0.338
	10	$<$0.001	$<$0.001	0.001	0.002	0.559
ML10M	0 (ISGD)	**0.007**	0.026	0.040	0.061	**0.176**
	1	**0.007**	0.024	0.039	0.065	0.241
	2	0.006	0.023	0.039	0.066	0.228
	3	0.006	0.023	0.040	0.070	0.292
	5	**0.007**	**0.027**	**0.048**	**0.083**	0.342
	10	0.001	0.003	0.006	0.013	0.468

Table 5.4: Aggregated results of RAISGD with $l \in \{0, 1, 2, 3, 5, 10\}$. With $l = 0$ RAISGD is equivalent to ISGD. Time is the average model update time in milliseconds. Best results in bold.

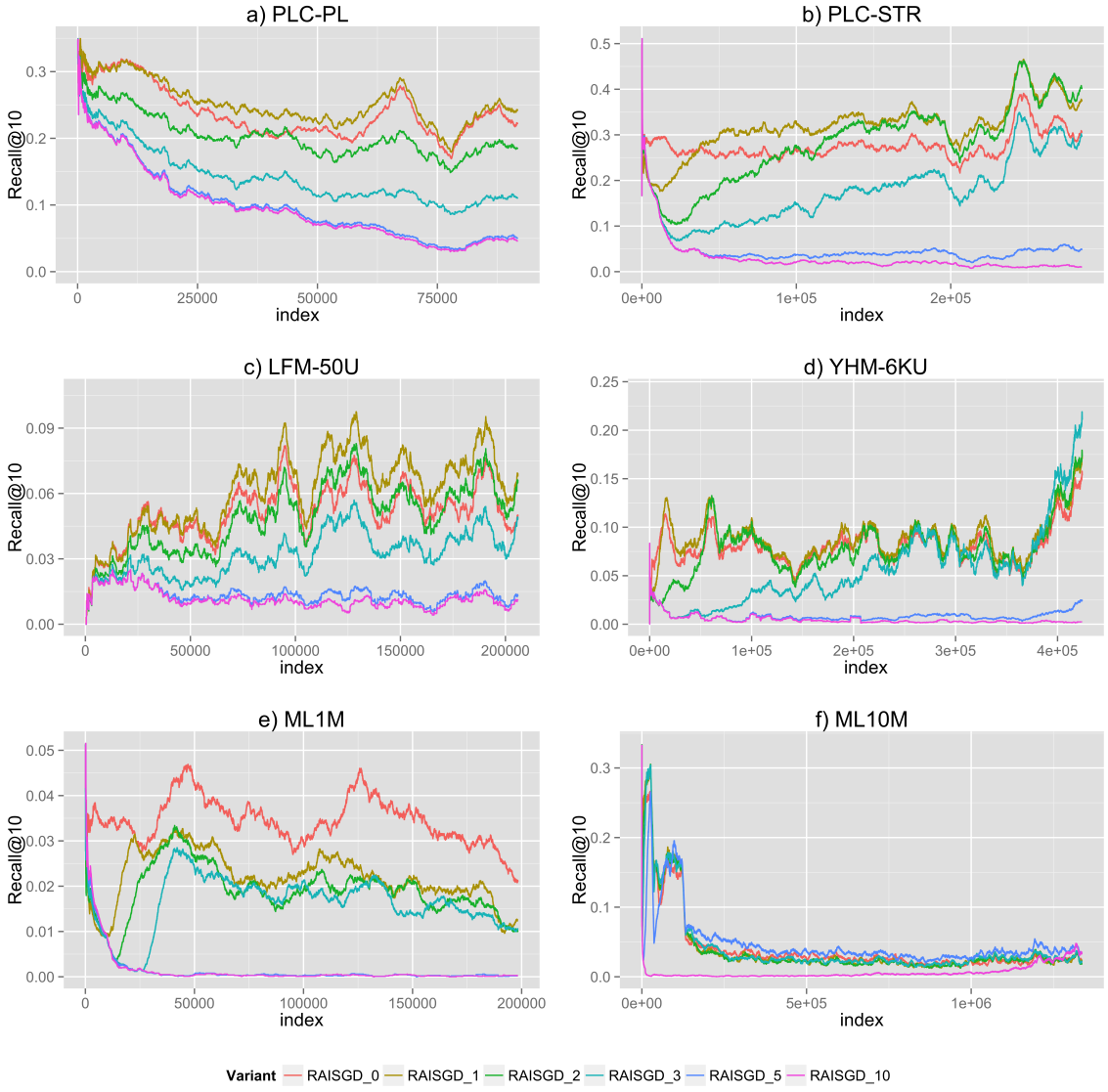


Figure 5.4: Prequential evaluation of Recall@10 with 6 datasets for RAISGD with $l \in \{0, 1, 2, 3, 5, 10\}$. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

average processing time does not increase in a high proportion. We calculate the increase of processing time in percentage of the original ISGD time in Table 5.5. The largest absolute increase in average update times is 1.245 ms and occurs with the YHM-6KU dataset with $l = 10$. This corresponds to a 28% increase on the average time required to perform an update without the overhead negative feedback. In terms of percentage of the original time with $l = 0$, the worst case is ML1M with $l = 10$, with a 710% increase. However, this corresponds to an absolute overhead of less that half a millisecond. In any case, the average update time never goes near 10ms, and the largest increases in percentage

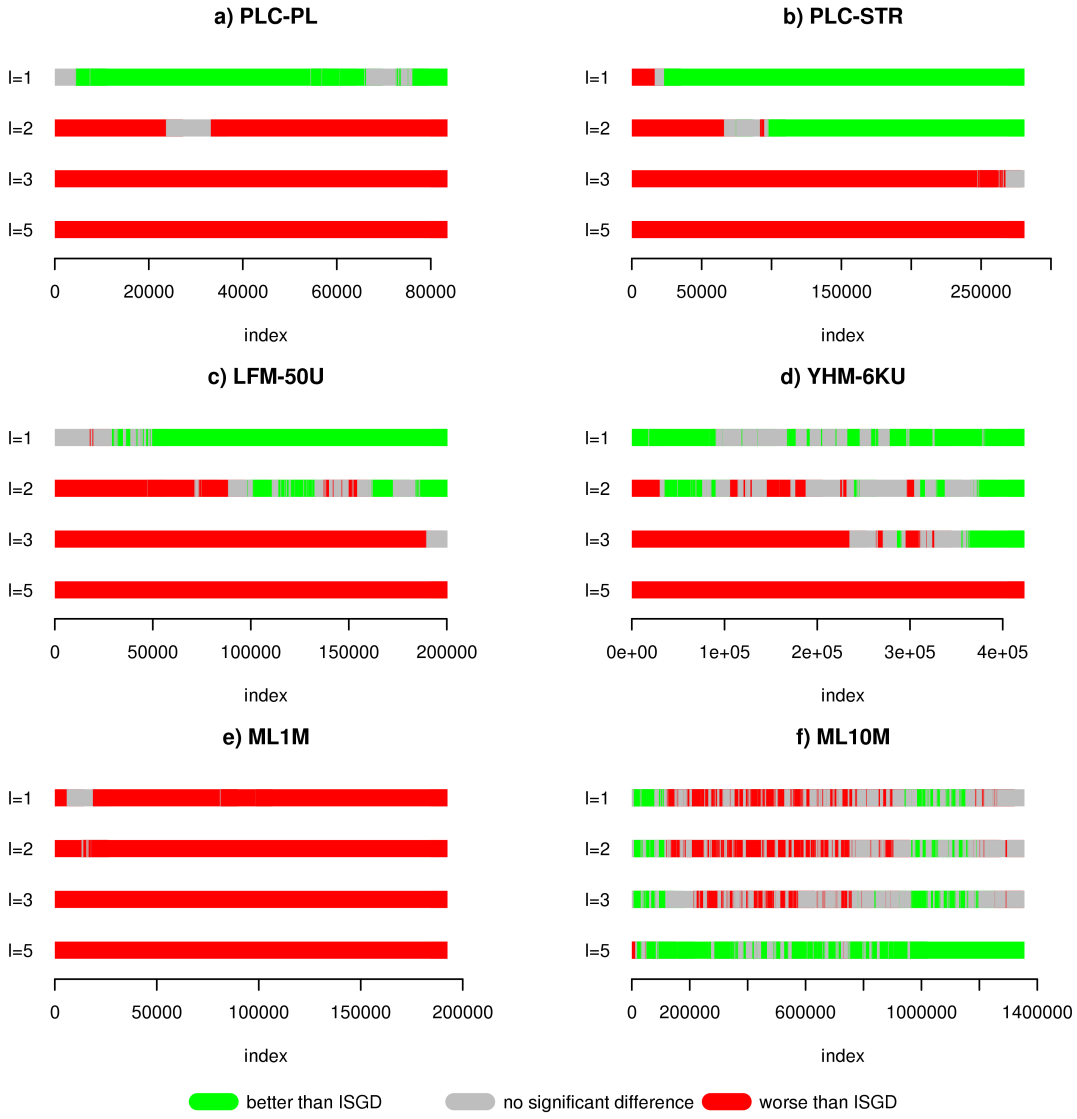


Figure 5.5: Signed McNemar pairwise test between Recall@10 obtained by RAISGD with $l \in \{1, 2, 3, 5\}$ with respect to ISGD (RAISGD with $l = 0$). The color of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that the corresponding value of l is significantly better, without significant difference, or significantly worse than $l = 0$, with a confidence level of 99%.

correspond to the datasets with the lowest absolute times, that do not even amount to 1ms.

Figure 5.6 shows a moving average of the actual update times at each step.

One interesting observation that can be done by looking at Figure 5.6 is that some datasets exhibit more variability in the update times than others. Clearly, the datasets with which the algorithms are slower, also show a high variability over time. It also becomes more clear

Dataset	Time ($l = 0$)	l	Time	Increase	Increase (%)
PLC-PL	0.535	1	0.586	0.051	10%
		2	0.714	0.179	33%
		3	0.754	0.219	41%
		5	0.809	0.274	51%
		10	0.949	0.414	77%
PLC-STR	0.237	1	0.273	0.036	15%
		2	0.379	0.142	60%
		3	0.414	0.177	75%
		5	0.483	0.246	104%
		10	0.655	0.418	176%
LFM-50U	2.625	1	2.333	-0.292	-11%
		2	3.008	0.383	15%
		3	3.017	0.392	15%
		5	3.069	0.444	17%
		10	3.185	0.560	21%
YHM-6KU	4.462	1	4.591	0.129	3%
		2	5.181	0.719	16%
		3	5.257	0.795	18%
		5	5.376	0.914	20%
		10	5.707	1.245	28%
ML1M	0.069	1	0.101	0.032	46%
		2	0.203	0.134	194%
		3	0.248	0.179	259%
		5	0.338	0.269	390%
		10	0.559	0.490	710%
ML10M	0.176	1	0.241	0.065	37%
		2	0.228	0.052	30%
		3	0.292	0.116	66%
		5	0.342	0.166	94%
		10	0.468	0.292	166%

Table 5.5: Differences of update times between RAISGD with $l \in \{1, 2, 3, 5, 10\}$ and $l = 0$, in milliseconds.

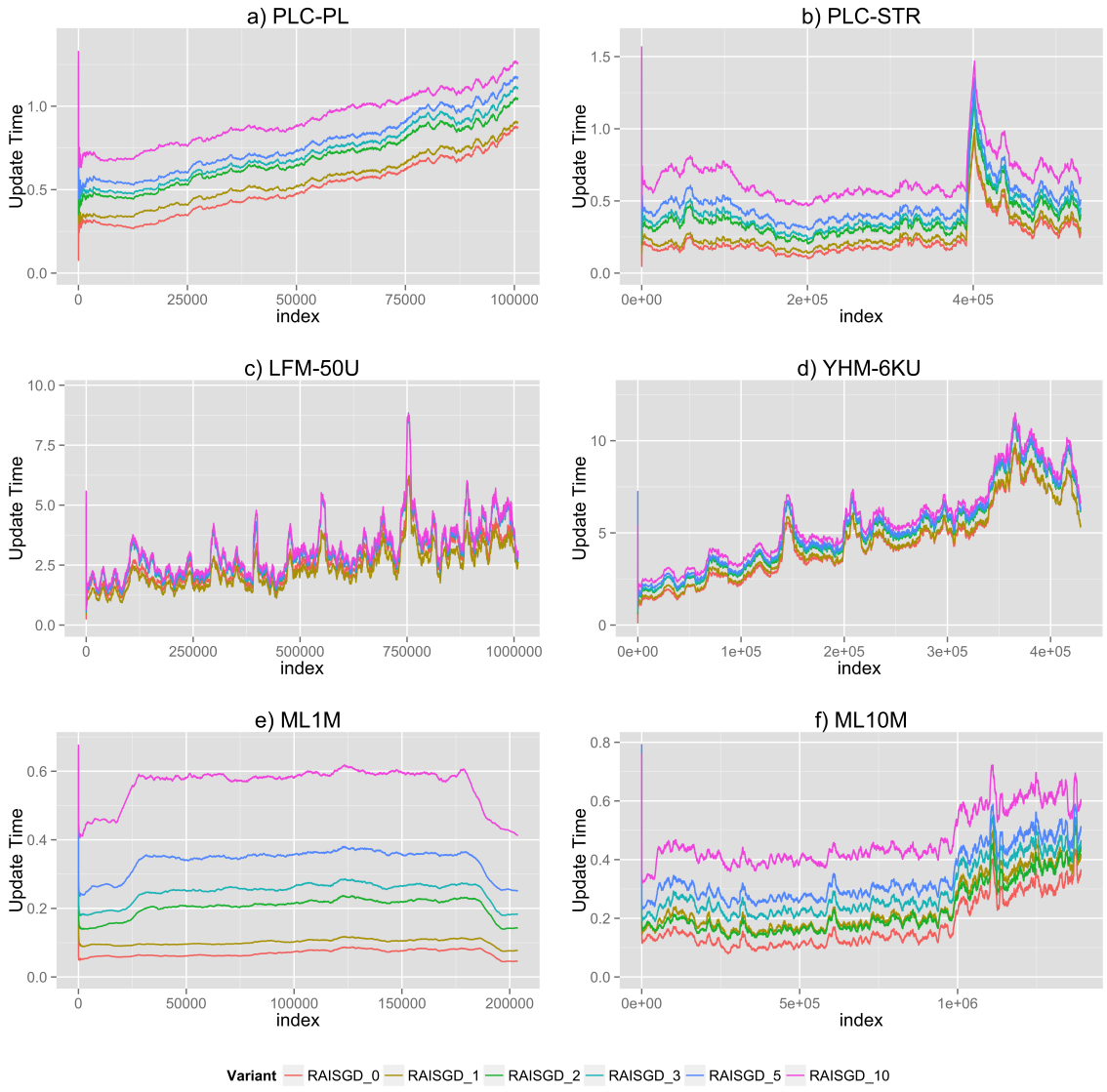


Figure 5.6: Online update times in milliseconds of RAISGD with 6 datasets and $l \in \{0, 1, 2, 3, 5, 10\}$, depicted with a moving average with $n = 10000$.

that these datasets are the ones for which l has the lowest impact. For most datasets, especially PLC-PL, LFM-50U and YHM-6KU, the update time clearly increases over time.

5.7.2 Rate-based negative feedback

In Section 4.3.2, we present an alternative method – RAISGD-RB (Algorithm 4.5) – to select negative feedback, that takes into account not only the recency of occurrence of items, but also their frequencies. In Figure 5.7 we depict the Recall@10 of both RAISGD

and RAISGD-RB with the six datasets, using the exact same parameters. The lines are almost indistinguishable, and the online McNemar test in Figure 5.8 indicates that there is no significant difference in most of regions, regardless of the dataset. This suggests that there is no considerable advantage in using this method. Given this similarity between the algorithms, and the added complexity of RAISGD-RB, we present the remaining results in this thesis with RAISGD only.

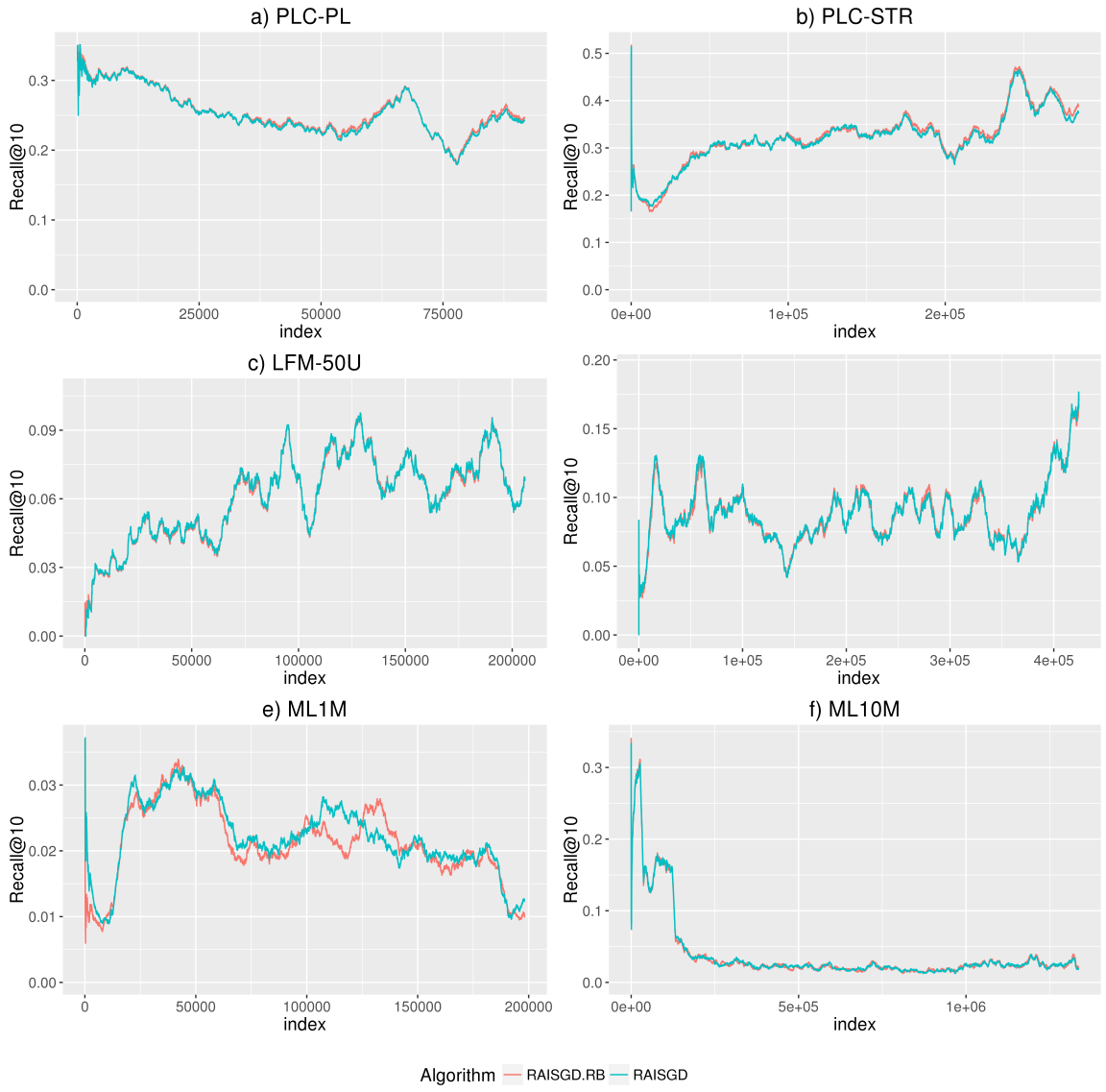


Figure 5.7: Prequential evaluation of Recall@10 with 6 datasets for RAISGD and RAISGD-RB with the same negative feedback amount l. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

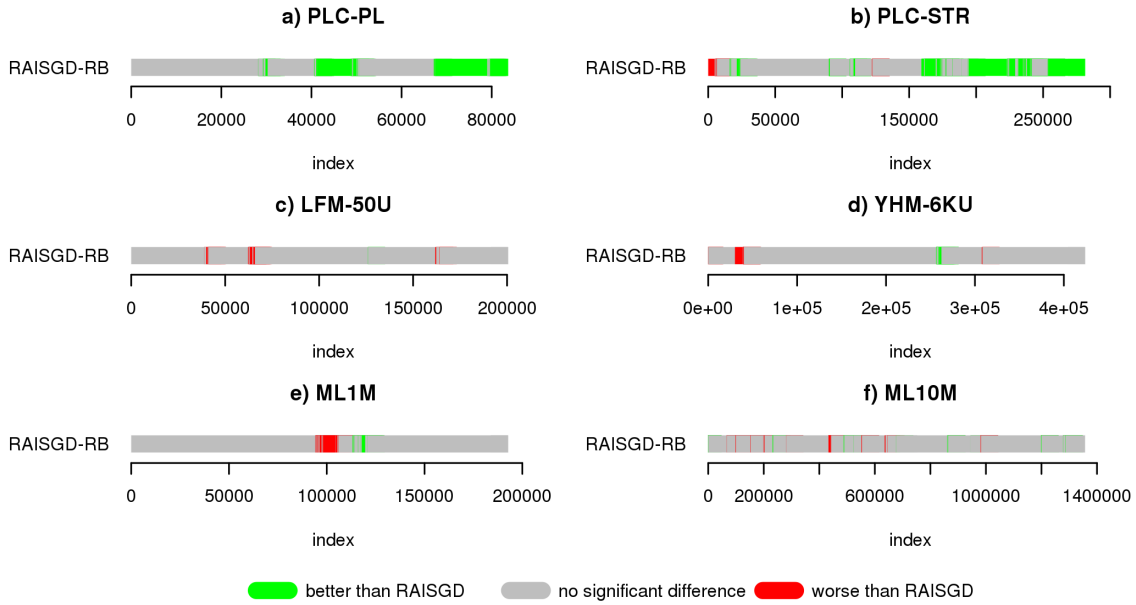


Figure 5.8: Signed McNemar pairwise test between Recall@10 obtained by RAISGD with respect to RAISGD-RB (with the same l). The color of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that the corresponding value of l is significantly better, without significant difference, or significantly worse than $l = 0$, with a confidence level of 99%.

5.7.3 Discussion

Results in Table 5.4 and Figure 5.4 indicate that using recency-based negative feedback is beneficial in most cases. The only exceptions may be the Movielens datasets ML1M and ML10M. With ML1M, negative feedback is hurtful for accuracy, with a significant degradation of recommendations. With ML10M, there is no significance difference between $l = 0$ and $l \in \{2, 3, 5\}$. This is confirmed by significance tests shown in Figure 5.5.

Another observation is that using too much negative feedback, especially with $l > 3$ generally causes significant degradation of Recall. The algorithm is more resilient to high values of l with the ML10M, where a setting of $l = 5$ slightly benefits accuracy. With $l = 10$, RAISGD fails to achieve acceptable performance.

The low time overhead of increasing the parameter l – the number of negative (u, i) pairs for each positive pair – can be explained by the fact that the atomic update operations are simple arithmetic calculations, with constant time complexity and usually correspond to a low proportion of the total time to update the model. Implementation and hardware issues related to random access to large data structures may dominate the update time. This

would also explain why update times tend to increase over time, as data structures become more and more populated.

Regarding rate-based negative feedback RAISGD-RB, our observations indicate that considering the frequency of items – besides recency – does not considerably improve the model.

5.8 Comparison with other algorithms

Our third and most comprehensive set of experiments, is intended to compare the accuracy of ISGD and RAISGD with a reference neighborhood-based algorithm – UKNN, Algorithm 3.1/3.2, and two different versions of a state-of-the-art factorization algorithm – BPRMF and WBPRMF, Algorithm 3.4. All the algorithms learn from a stream of positive-only user-feedback.

5.8.1 Optimal parameters

We use the methodology described in Section 5.4.5 to find the optimal settings for data dependent hyperparameters used in every algorithm. Table 5.6 summarizes these settings. ISGD and RAISGD share the same settings except for l, which is number of negative user-item pairs for each positive observation in RAISGD. ISGD is RAISGD with $l = 0$. Note that the optimal l for ML1M is 0 – i.e. no negative feedback. In the experiments, we still use $l = 1$ for RAISGD with ML1M, to allow a comparison with ISGD.

Another important note is that the optimal value for l with ML10M found using the methodology described in Section 5.4.5 does not correspond to the best performing setting observed in Section 5.7. This may happen because the initial 10% segment of the data exhibits a behavior that is very different from the remaining of the dataset (see Figures 5.4 d) or 5.9 d)). The setting found using our methodology could be appropriate for that initial segment, but not ideal for the rest of the data. Even taking this into account, Figure 5.5 d) suggests that the differences between different values of l are not very significant. We use $l = 2$ for this case, simply to stick with the same methodology in all experiments.

For UKNN, we use the same setting for number of neighbors $k = 10$, regardless of the dataset. The main objective of using UKNN is to establish as a reference for comparison, given that it is a well studied algorithm. We assume that setting of $k = 10$ is enough to have a valid enough reference.

Finally, we estimate the optimal values for all hyperparameters of BPRMF and WBPRMF,

Dataset	Algorithm	f	$iter$	η	λ	λ_u	λ_i	l
PLC-PL	(RA)ISGD	90	8	0.3	0.4			1
	BPRMF	100	4	0.1		0.15	0.15	
	WBPRMF	120	35	0.55		0.01	0.01	
PLC-STR	(RA)ISGD	200	6	0.35	0.50			1
	BPRMF	180	7	0.5		0.001	0.005	
	WBPRMF	200	50	0.5		0.08	0	
LFM-50U	(RA)ISGD	160	4	0.5	0.40			1
	BPRMF	140	45	0.2		0.15	0	
	WBPRMF	100	10	0.3		0	0	
YHM-6KU	(RA)ISGD	200	9	0.25	0.45			1
	BPRMF	100	3	0.2		0.001	0.05	
	WBPRMF	80	20	0.2		0.2	0	
ML1M	(RA)ISGD	160	8	0.1	0.4			0
	BPRMF	100	25	0.3		0.01	0.01	
	WBPRMF	100	35	0.2		0.01	0.2	
ML10M	(RA)ISGD	70	10	0.15	0.45			2
	BPRMF	100	9	0.05		0.005	0.05	
	WBPRMF	100	20	0.15		0.01	0.05	

Table 5.6: Optimal parameter settings for ISGD, RAISGD, BPRMF and WBPRMF. ISGD and RAISGD share the same settings except l, which is a parameter for RAISGD only.

except λ_j (see Algorithm 3.4). In our experiments, we verified that the impact of changing λ_j was very low when compared with the other regularization factors. In all experiments, we set $\lambda_j = 0.00025$, which is the default value in the software package that implements the algorithms.

5.8.2 Results

We run all algorithms with the six datasets, training on the first 10% of the data, as described in Section 5.4, and collect results in Table 5.7. With four datasets – PLC-PL, PLC-STR, LFM-50U and YHM-6KU – both RAISGD and ISGD outperform BPRMF, WBPRMF and UKNN regarding Recall at all cutoffs.

Regarding the average model update time, ISGD and RAISGD are clearly faster than the other tested algorithms. Expectedly, UKNN is by far the most computationally demanding algorithm. However, looking closely to the absolute time in milliseconds, it is safe to say that even though much more expensive, it is still usable with streaming data, if the rate

of the data is not extremely fast. The only case where UKNN gows beyond 1 second (in average) to update the model is with LFM-50U. The second worse case is with ML10M, with an average update time of approximately 178 milliseconds, which is manageable in many applications.

Table 5.7 shows the overal averages of recall and time for each datasets, but do not give any information on how recall and time vary over time. In Figure 5.9, a moving average with size $n = 10000$ depicts the evolution of Recall@10 during the experiments. For the sake of clarity, we place the plots for Recall@$\{1, 5, 20\}$ in Annex B. With this visualization, the dynamics of Recall become visible. For example, with LFM-50U, in Figure 5.9 c), ISGD and RAISGD tend to gradually improve as more data becomes available, and the improvement of RAISGD over time is higher than the improvement of ISGD. With PLC-PL and YHM-6KU, the accuracy of ISGD and RAISGD have roughly the same dynamics, with the lines corresponding to both algorithms having approximately the same shape and value.

The information in both Table 5.7 and Figure 5.9 gives a general perspective about the relative accuracy of algorithms, it is not very clear in some cases, if the differences are big enough to validate a claim that one algorithm really is better than other. One already mentioned example is RAISGD and ISGD with PLC-PL and YHM-6KU – Figure 5.9 a) and d) –, where Recall is very close or almost indistinguishable. To be able to safely draw conclusions on results, we perform statistical significance tests using the signed McNemar test over a sliding window, as described in Section 5.4.4. This test gives support to observations that are not very clear by just looking at numerical results or plots. Because McNemar is a pairwise test, a complete comparative assessment with six datasets, five algorithms and four metrics would require a total of 240 tests. Given that in this thesis we are mainly interested in assessing the relative performance of RAISGD, we present pairwise comparisons between RAISGD and every other algorithm, in a total of 24 tests based on Recall@10. These are depicted in Figure 5.10 (plots based on Recall@$\{1, 5, 20\}$ are available in Annex B). We use a sliding window with size $n = 10000$, same as the moving average window used in Figure 5.9. The visualization in Figure 5.10 provides information about the significance of differences between RAISGD and all other algorithms evolve over time. In other words, it tells us when RAISGD it is *significanlty* better, not *significantly* different and *significantly* worse than all other algorithms. In the particular cases evaluated – RAISGD against all others –, the McNemar test essentially shows that the differences that are seen with clarity in Figure 5.9 are statistically significant.

Similarly to Recall, we also depict the evolution of the time required to update the model in Figure 5.11. We leave out the line relative to UKNN, given that it would squash the scale down to the point where differences between the other algorithms would eventually become impossible to perceive. Figure 5.11, besides confirming the measurements presented in Table 5.7, provide some extra information. For example, BPRMF and WBPRMF generally

Dataset	Algorithm	Recall@1	Recall@5	Recall@10	Recall@20	Upd. (ms)
PLC-PL	UKNN	0.028	0.077	0.107	0.138	11.963
	BPRMF	<0.001	0.002	0.002	0.005	1.502
	WBPRMF	0.010	0.030	0.043	0.059	1.693
	ISGD	0.104	0.199	0.234	0.265	**0.535**
	RAISGD	**0.115**	**0.211**	**0.249**	**0.282**	0.586
PLC-STR	UKNN	0.023	0.065	0.094	0.127	54.785
	BPRMF	0.001	0.007	0.012	0.020	1.799
	WBPRMF	0.001	0.002	0.002	0.005	1.992
	ISGD	0.127	0.241	0.277	0.302	**0.237**
	RAISGD	**0.172**	**0.285**	**0.322**	**0.353**	0.273
LFM-50U	UKNN	<0.001	<0.001	0.001	0.002	>1000
	BPRMF	<0.001	<0.001	0.001	0.001	229.820
	WBPRMF	<0.001	<0.001	<0.001	<0.001	225.545
	ISGD	0.034	0.049	0.052	0.055	2.625
	RAISGD	**0.044**	**0.058**	**0.062**	**0.065**	**2.333**
YHM-6KU	UKNN	0.007	0.026	0.043	0.065	110.868
	BPRMF	0.003	0.010	0.018	0.028	11.678
	WBPRMF	0.004	0.019	0.032	0.049	10.967
	ISGD	0.030	0.063	0.082	0.103	**4.462**
	RAISGD	**0.037**	**0.070**	**0.090**	**0.112**	4.591
ML1M	UKNN	**0.015**	**0.058**	**0.096**	**0.152**	6.533
	BPRMF	0.008	0.044	0.079	0.135	0.242
	WBPRMF	0.008	0.037	0.067	0.118	0.236
	ISGD	0.005	0.021	0.034	0.055	**0.069**
	RAISGD	0.003	0.013	0.022	0.039	0.101
ML10M	UKNN	**0.010**	**0.036**	**0.059**	**0.093**	177.876
	BPRMF	**0.010**	0.035	0.056	0.076	0.421
	WBPRMF	0.009	0.037	0.056	0.076	0.348
	ISGD	0.007	0.026	0.040	0.061	**0.176**
	RAISGD	0.006	0.023	0.039	0.066	0.228

Table 5.7: Overall results with RAISGD, ISGD, BPRMF, WBPRMF and UKNN. Best performing algorithms are highlighted in bold for each dataset. Update times are the average value of the update time in milliseconds for all data points.

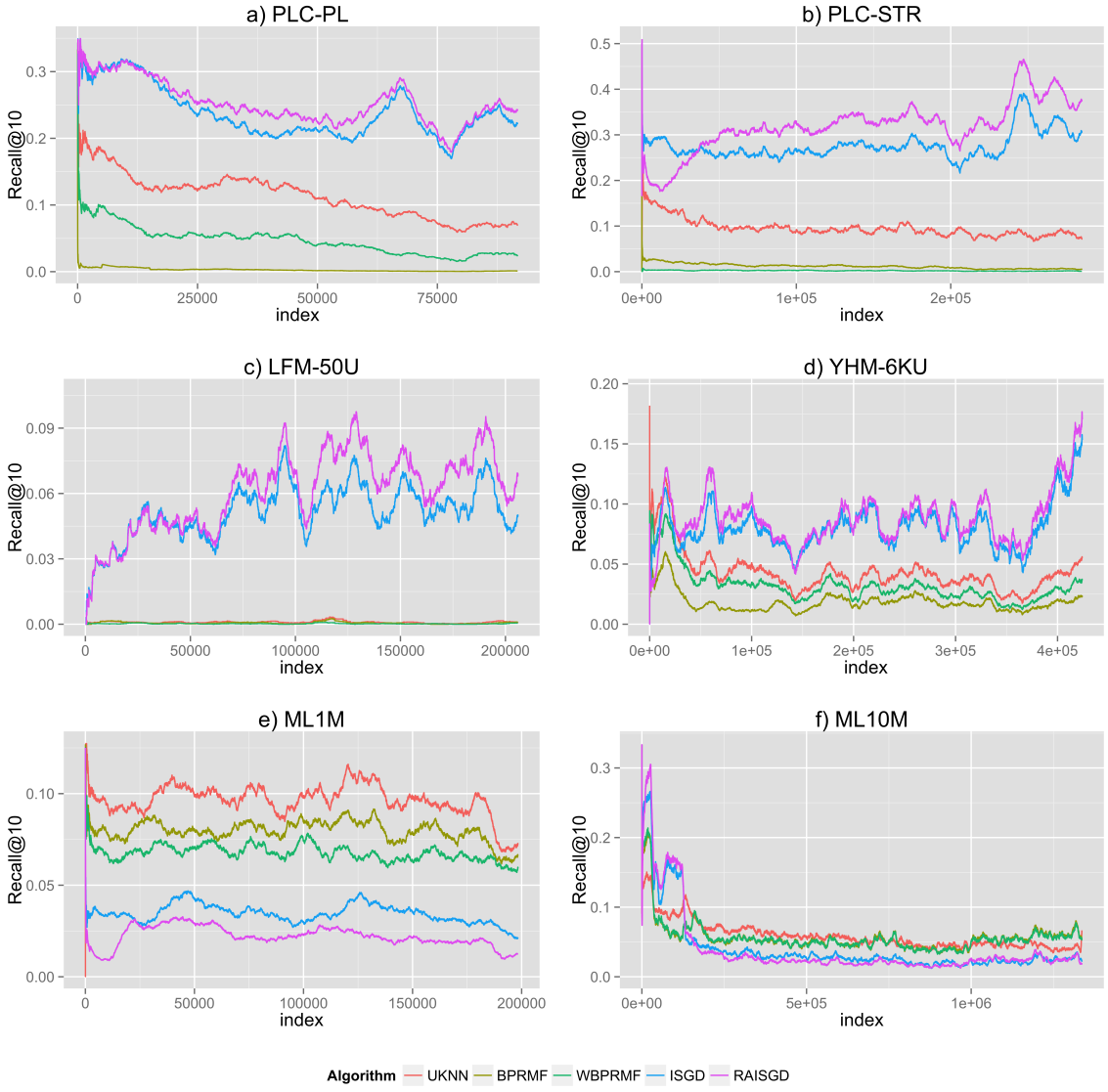


Figure 5.9: Prequential evaluation of Recall@10 with 6 datasets. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

require more time, but it is also visible that both these algorithms are a more unstable than ISGD and RAISGD in terms of processing time, except with ML10M – Figure 5.11 e).

5.8.3 Discussion

The comparison between several algorithms suggests that RAISGD performs better that both classic algorithms – UKNN – and state-of-the-art alternatives – (W)BPRMF. However,

Figure 5.10: McNemar pairwise tests between RAISGD and other algorithms, relative to the Recall@10 metric. The colour of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that RAISGD is significantly better, without significant difference, or significantly worse than the referenced algorithm, with a confidence level of 99%.

the evidence is limited to four of the six datasets with which we perform experiments. In two of the datasets – ML1M and ML10M –, RAISGD, as well as ISGD, perform significantly worse than all other algorithms. It may be natural that both ML1M and ML10M share some properties, given that they are taken from the same source, domain, and are both manipulated to simulate a positive-feedback stream – they originally contain movie ratings. This manipulation – that consists of filtering out ratings below a certain value – does

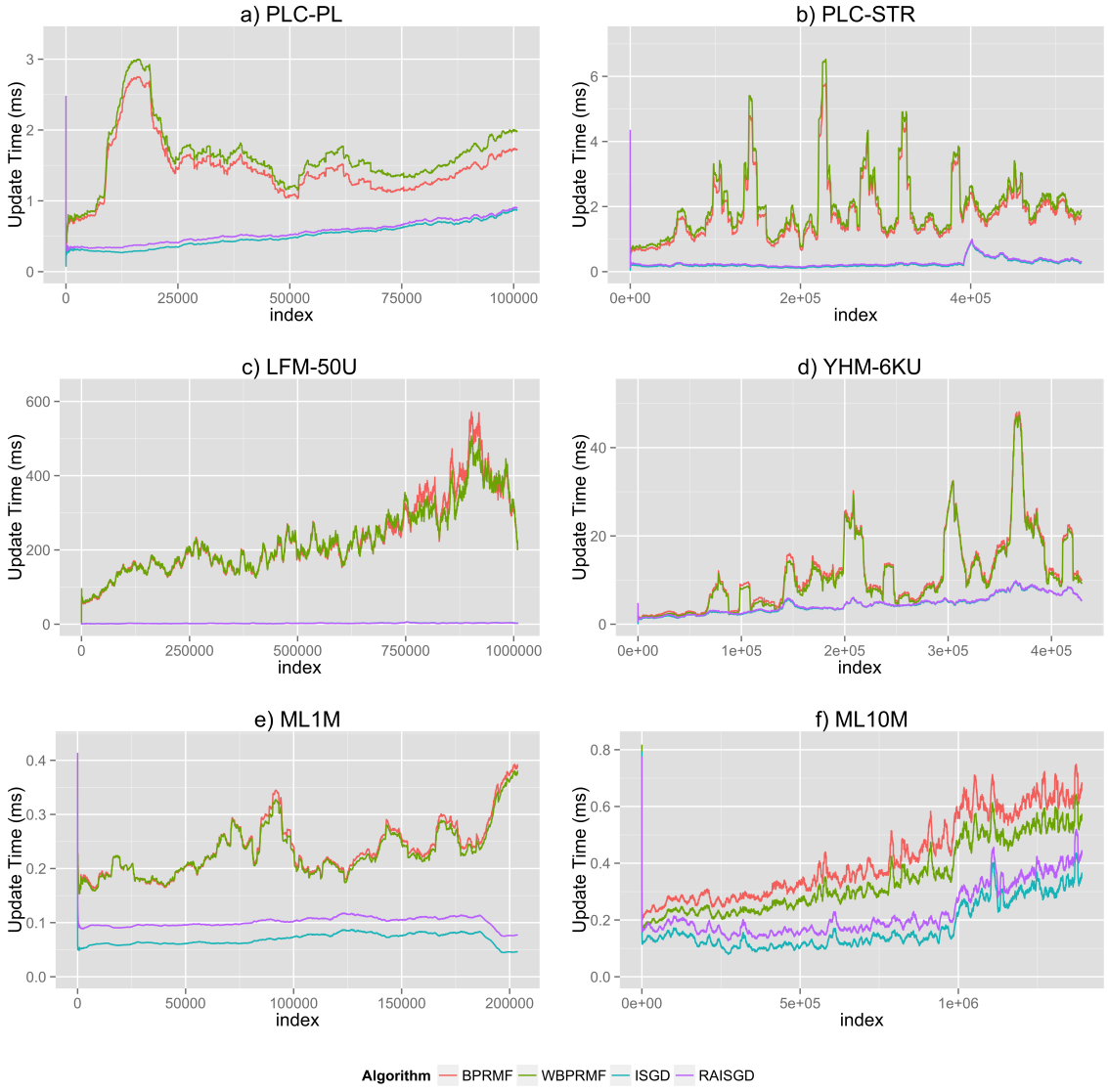


Figure 5.11: Online update times in milliseconds with 6 datasets, represented by a moving average with $n = 10000$.

not seem to explain the relative performance of algorithms. The same kind of filtering is performed with the YHM-6KU, and that does not seem to be a problem for RAISGD and ISGD with this dataset.

The measurements of incremental model update times is also a fundamental aspect to evaluate. We want algorithms that deal with data streams to be able to process data faster than its arrival rate. All tested algorithms comply with this constraint. With no surprise, UKNN is much more time consuming, given its much higher computational complexity. However, it still seems to yield average update times that may actually be applicable in many

real-world scenarios where the data rate is not extremely high. Between the ISGD and BPRMF variants, it is clear, that RAISD and ISGD are faster than BPRMF and WBPRMF in all cases, without exceptions.

ISGD is based on a matrix factorization method that uses Stochastic Gradient Descent to learn from very large ratings datasets. This is a widely used method in recommender systems. ISGD has two fundamental differences. First, the capability to maintain a recommendation model incrementally, enables the algorithm to timely process fast streams of user feedback, without the need to revisit past data, or to do periodic batch retraining. Second, the algorithm is designed to deal with positive-only data, which broadens its applicability in real-world applications. In our first set of experiments, it comes with no surprise that ISGD is much more efficient than its batch version, when dealing with streaming data. The main conclusion regarding this, is that to achieve a level of accuracy comparable to ISGD, the batch version of the algorithm needs much more time – several orders of magnitude more than the ISGD. To save processor time, by increasing the interval at which the batch algorithm is retrained, leads to severe accuracy loss.

Because ISGD is based on a method designed to process ratings, it has a fundamental limitation when dealing with positive-only data. The recommendation model essentially needs to be able to distinguish between good and bad recommendation candidates. The algorithm needs to learn this exclusively from the positive examples – the only ones available. One workaround is to use a learning-to-rank approach, that directly models a ranking of items for each user. Another alternative is to artificially introduce negative feedback in the data. Some schemes to chose negative examples are available in the literature, however they require batch processing of data and are not applicable in a streaming environment. We propose a recency-based scheme that introduces negative feedback in the stream using the least recent items – the items that occurred longer ago. In our second set of experiments, we measure the impact of this technique. We use several amounts of negative feedback, from 1 to 10 negative examples for each (positive) example in the stream. We conclude that, in most cases, negative feedback significantly improves accuracy. Furthermore, this improvement in accuracy does not come with a high cost in the time needed to process the extra feedback. Even with 10 negative examples for each example, the absolute time to update the model does not increase more than a few milliseconds.

In terms of accuracy, when compared with other algorithms, ISGD beats both classic and state-of-the-art approaches in four datasets, and looses in two other datasets. With respect to time, ISGD is clearly faster, including when using negative feedback – RAISGD. It is safe to conclude that ISGD and RAISGD are at least competitive in accuracy and extremely fast. This is especially important when the rate of the data is very high, which can happen easily in large-scale systems. Our main contribution – the algorithms – is therefore valuable for a large number of applications. The competitive accuracy with a considerable variety

of datasets and the low update times allow us to safely state that our contribution is a good choice for a large number of Recommender Systems, especially those that need to continually maintain up-to-date models.

We note that the ML10M dataset exhibits very abrupt degradation in Recall after the initial stage of the prequential process, then stabilizing at a relatively low Recall. Since this behavior is common to all algorithms and/or techniques, we did not investigate further the causes of this accident. One possible explanation is that a global drift may be present in the dataset that would require the re-adjustment of hyperparameters – these are optimized using only the first 10% of the available data, and may not be optimal for the entire dataset.

Regarding the methodological aspects of the evaluation of recommender systems, we propose a protocol based on prequential evaluation, that consists of using each available example to perform both evaluation of recommendations and learning continuously. We argue that recommender systems, both offline and online, would greatly benefit from integrating the prequential evaluation process in the system. This allows continuous monitoring of the system, as well as facilitates online testing with real users.

5.9 Summary

One crucial aspect of the research in recommender systems is evaluation. The majority of the literature focuses on off-line accuracy and scalability evaluation using well studied evaluation protocols and metrics. However, production systems are usually sensitive to a large number of environmental variables that cannot be reproduced in the laboratory. Users of online systems are humans with naturally biased perspectives on the quality and the utility of a recommender [McNee et al., 2006]. Moreover, the main task of a recommender system may vary considerable with the application [Herlocker et al., 2004]. For instance, users may be willing to sacrifice accuracy to obtain serendipitous, less obvious recommendations. In other applications, such as news recommendation, the recency of recommended items is a key factor and may be preferred to high topic relevance. On the other hand, scalability issues typically have considerable investment implications. This type of factors cause algorithms with good off-line performance to not translate directly into good online performance. For this reason, online evaluation and user feedback may be determinant to the choice of algorithms and their parameters. One practical way of evaluating online recommenders is by conducting controlled experiments [Kohavi et al., 2009] involving real users in a live environment, however this is not always possible, since access to production-level recommender systems is not easily available.

In this chapter we have proposed the adoption of prequential evaluation for recommender systems, which is extremely useful to evaluate recommender systems that deal with data

streams. Using this evaluation framework, it is possible to monitor and continuously evaluate online recommender systems. Moreover, while not being a substitute for for either online A/B testing and user surveys, or offline benchmarking, it can be used as a powerful tool to facilitate both evaluation environments – online and offline.

We have conducted a series of experiments to assess the validity of our claims, namely that:

1. the incremental approach is a good solution to process user generated data online;
2. online bagging improves the accuracy of ISGD;
3. using recency-based negative feedback imputation helps incremental algorithms that process positive-only data;
4. the proposed solutions (ISGD and RAISGD) are highly competitive with state-of-the-art methods in streaming environments.

Our results suggest the validity of all three claims, however with some considerations. First, it is no surprise that claim 1. is validated. Obviously, incremental algorithms are much more efficient in processing data streams. Second, claim 2. is not validated by *all* experiments. In one of the six datasets, using negative feedback was actually hurtful for the algorithm's accuracy. Regarding claim 3, we show that online bagging can improve the accuracy of ISGD, but with a considerable time overhead when computing recommendations. The fourth claim is verified in four of the six datasets, with our proposed algorithms being outperformed both UKNN and (W)BPRMF. Regarding time, ISGD is the fastest algorithm in practically all cases, followed by RAISGD, which is naturally more time consuming. Nevertheless, in all cases, both ISGD and RAISGD outperform the (W)BPRMF alternatives by a considerable amount of time. UKNN is by far the worse algorithm in terms of time requirements, however it may still be usable in most streaming applications given that it takes less than a second to perform incremental updates in five out of six datasets.

Chapter 6

Conclusions

In this thesis, we have identified several problems that arise from the typical batch approach to recommendation problems. Algorithms that process data in batch usually disregard the dynamic nature of the process that generates user feedback – the online user activity – and treats datasets as monolithic instances of the problem. As a result, time evolving concepts, such as user preferences, global trends and introduction/removal of users and items are not correctly captured. Additionally, operational and computational constraints tend to increase, as the amount of available data quickly and continuously increases.

We have found that the existing contributions in the field of recommender systems do not adequately address these issues. While valuable contributions are available on algorithms that are sensitive to time, and also in the scalability of algorithms, the fundamental problems are yet to be fully solved. Our goal in this thesis has been to address these problems. To do that, we have formulated four research questions and have provided four contributions to address them. In the following sections we specify our contributions and conclusions regarding each research question. For convenience, we replicate below the research questions formulated in Chapter 1 in the beginning of each Section below.

6.1 The impact of time

RQ1 *Do phenomena related with time have a significant impact on recommendation? If so, is the existing knowledge in the field of recommendation sufficient to approach time related problems?*

We have reviewed the most relevant literature on state-of-the-art recommendation algorithms, focusing particularly on contributions that deal with the time dimension. We have divided these contributions in two categories, depending on how time is approached. The

first category consists of time-aware algorithms, that explicitly deal with the measured time, either using timestamps or other time related information available in the data. The second category, time-dependent algorithms, encompasses the algorithms that are sensitive to time even though learning is not performed with explicit time features. Instead, the algorithms are sensitive to the natural sequence of events in the data and do no require data to carry explicit time-related features.

We have concluded that the exploitation of time is beneficial to the accuracy of algorithms. However we have detected that most of the proposed algorithms have much higher runtime complexity than their time agnostic counterparts. Furthermore, most of the reviewed contributions not only consist of batch – with the inherent scalability limitations – but also have higher runtime complexity than their time-agnostic counterparts. Therefore, we have concluded that the currently available literature does not fully cover the problems with learning time sensitive recommendation models.

6.2 The data stream approach

RQ2 *Is the batch learning approach adequate for online, real world, recommender systems? Could the techniques and algorithms for data streams be used to improve the accuracy and/or scalability of recommender systems??*

We have argued that batch learning is not adequate for online systems where data is being continuously generated. To address this, we have approached the recommendation problem as a data stream problem, and investigated the state-of-the-art on incremental algorithms for recommendation, able to easily incorporate new data in the model, and forgetting mechanisms, that enable the model to forget outdated or irrelevant information.

We have proposed a fast incremental matrix factorization algorithm for recommender systems – ISGD –, able to learn from online streams of positive-only user feedback data. We have shown that the incremental algorithm yields much higher accuracy than a similar batch alternative. Moreover, to achieve comparable accuracy, the batch algorithm needs to be retrained with a frequency that is prohibitively high – given its high processing time – in realistic scenarios. Our results show that the ISGD significantly outperforms both state-of-the-art and classic incremental algorithms on most datasets, in terms of predictive ability and in terms of running times. We also show that ISGD benefits from the use of online ensemble techniques. Our evaluation of ISGD with Bagging shows that the algorithm's accuracy improves considerably.

6.3 Learning from positive-only ratings

RQ3 *Considering the known problems of dealing with positive-only data for recommendation, can we devise a scheme that mitigates those problems that is compatible with the streaming approach? Can we exploit the time dimension to do this?*

To overcome the challenges of learning from positive-only data, we have devised two recency-based schemes that artificially introduce negative examples in the stream, based on the recency of occurrence of items and their frequency in the stream. To evaluate the resulting algorithms – RAISGD and RAISGD-RB – we have followed several experiments. In a set of experiments, we have measured the impact of using the recency-based negative feedback imputation, by comparing RAISGD with the baseline ISGD. Our results show that using this recency-based scheme, the accuracy of the incremental algorithm significantly increases. Obviously, given that ISGD is already competitive with other methods, RAISGD improves over this. Furthermore, even with the time overhead to perform the additional learning steps – required to learn from artificial negative examples –, RAISGD still outperforms state-of-the-art algorithms in running time. RAISGD-RB, that uses both the recency and the frequency of occurrence of items does not improve over RAISGD, which only accounts for the recency. We conclude that it is beneficial to use time related information to address the problem of learning recommendation models exclusively from positive examples.

6.4 Evaluating stream-based recommenders

RQ4 *Are traditional evaluation protocols suitable for recommender systems running on dynamic real-world environments? If not, how de we evaluate recommendation algorithms in such environments?*

Classic evaluation methodologies to evaluate batch recommendation algorithms are not directly applicable to algorithms that learn incrementally from data streams. In our literature review, except for our own contributions we have found very few others that use an evaluation methodology adequate for incremental models. We have contributed with an evaluation framework that uses the prequential method to continuously evaluate recommender systems as they operate incrementally, applicable to both real world and simulated environments. We have successfully used this methodology to compare our incremental matrix algorithms – with and without negative feedback imputation – with a classic neighborhood-based reference algorithm and two versions of a state-of-the-art learning-to-rank algorithm. Based on our findings we conclude both algorithms are superior both in terms of accuracy

and processing speed in most situations. The continuous evaluation process allows not only to perform offline experiments with commonly used metrics, but also to continuously monitor the evolution of those metrics in a simulated or online environment, where data keeps flowing in. It is also possible to make reliable comparisons with significance tests by collecting online statistics of the outcome of the learning process, and providing clear visualizations of the learning process' outcome.

6.5 Limitations

Negative feedback definitely improves the accuracy of algorithms in most evaluated problems – only in one of six datasets it is harmful. A useful feature would be to compute the optimal amount of negative feedback online. We did not find a useful relation between any property of the datasets and the optimal amount of negative feedback. Although some exploratory experiments regarding this potential feature were conducted, they did not provide useful insights.

A quick observation of the results of our experiments, reveals that our proposed method does not outperform all other algorithms when running with all datasets. Specifically with two of the six datasets – both from Movielens [Grouplens, 2013] –, classic and state-of-the-art algorithms achieve similar or even slightly better performance. The Movielens datasets are widely used in the evaluation of recommender systems, in the literature. Not being able to achieve significantly better performance with these datasets is obviously undesirable. However, since that on all other datasets our algorithms clearly outperform the alternatives, this may also indicate that the Movielens datasets represent a very particular case of the recommendation problem.

Finally, during the period of the work for this thesis, we did not have a chance to evaluate our work in a production system, with real users. It is well known that to conduct these types of experiments requires a well established industry partner willing to share resources and also a reasonably large time window. Although these two conditions were not met during the period of the project, we expect to be able to perform online experiments in the near future.

6.6 Future work

A large part of our future work is to address the issues above. Specifically, we need to understand why algorithms do not perform as well with some few datasets as they do with most others. For example, it is clear that with the datasets extracted from the Movielens data

[Grouplens, 2013], the methods proposed in this thesis are only beneficial for the account of speed. Another issue is how to find good estimators for the algorithms' hyperparameters online. For that, it is necessary to establish clear correlations between properties of the data and those hyperparameters, which can be a challenging task. One way to address this problem is to use meta-learning [Brazdil et al., 2009]. This is a discipline in the field of machine learning, that learns models able to choose the best algorithms and parameters for each dataset. A challenging issue here would be how to use such a framework with data streams [van Rijn et al., 2014].

A related research direction is to be able to explicitly detect changes and timely react to them. These changes may include, in the case of recommender systems, long-term preference changes, sudden preference shifts, global trends, seasonal effects, and many other phenomena, depending on the application. Once a change is detected, action can be taken, such as tuning a parameter, switching between algorithms, or applying different output filters.

Regarding negative feedback imputation, more sophisticated models to choose negative examples can be investigated. The recency-based scheme is applied globally, one option would be to use a personalized – user by user – recency-based scheme, with good results. It would also be interesting to combine negative feedback imputation with forgetting mechanisms, which we have not done yet. We plan to combine these two lines of work in the near future.

Appendix A

List of Abbreviations

ALS - Alternating Least Squares

CF - Collaborative Filtering

CP - CANDECOMP/PARAFAC

LSA - Latent Semantic Analysis

LSI - Latent Semantic Indexing

MF - Matrix Factorization

OCCF - One-Class Collaborative Filtering

POI - Point-Of-Interest

RS - Recommender Systems

SGD - Stochastic Gradient Descent

SVD - Singular Value Decomposition

Appendix B

Additional plots

B.1 ISGD with bagging

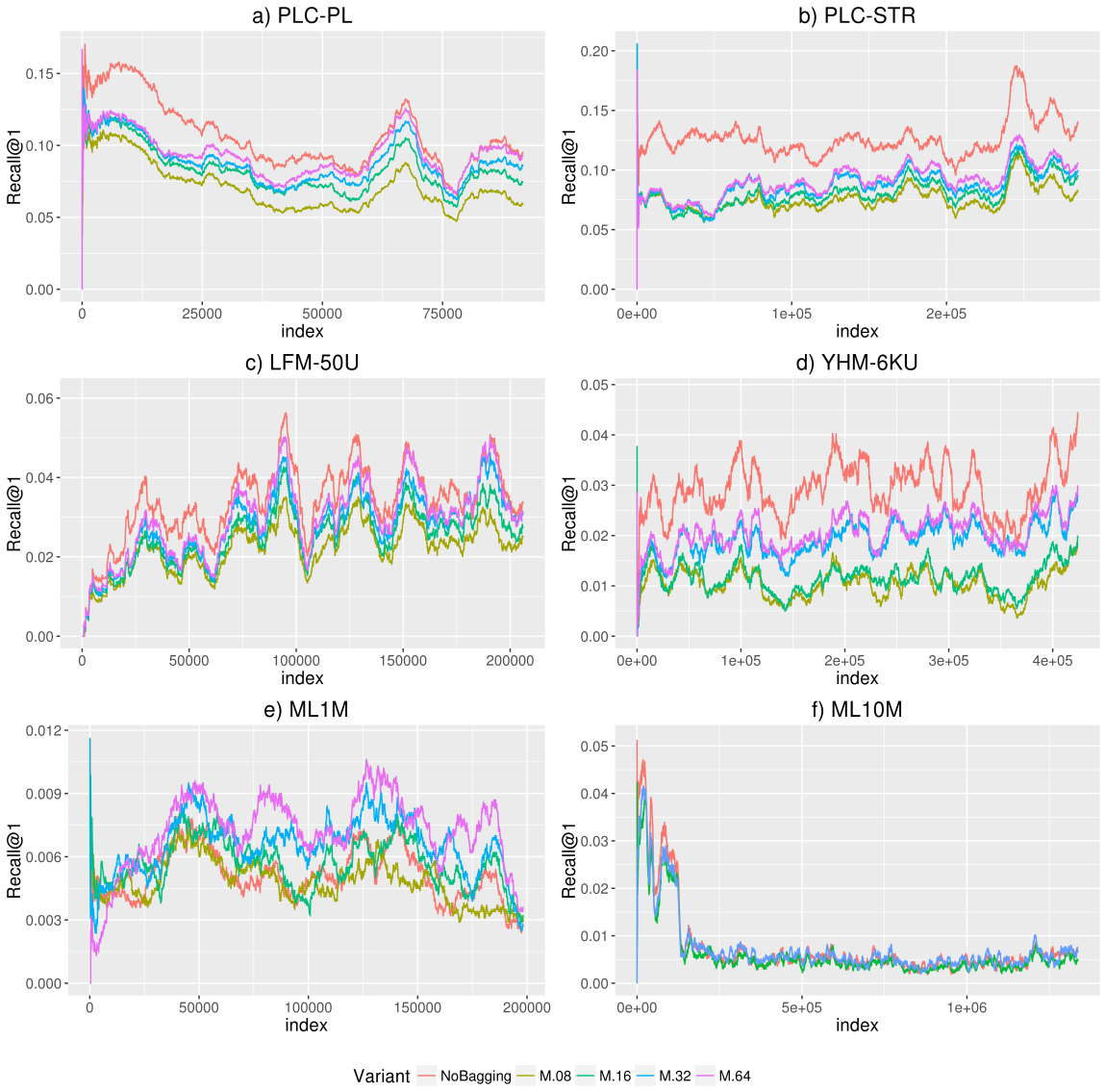


Figure B.1: Prequential evaluation of Recall@1 with ISGD with and without bagging. Lines are drawn using a moving average of Recall@1 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

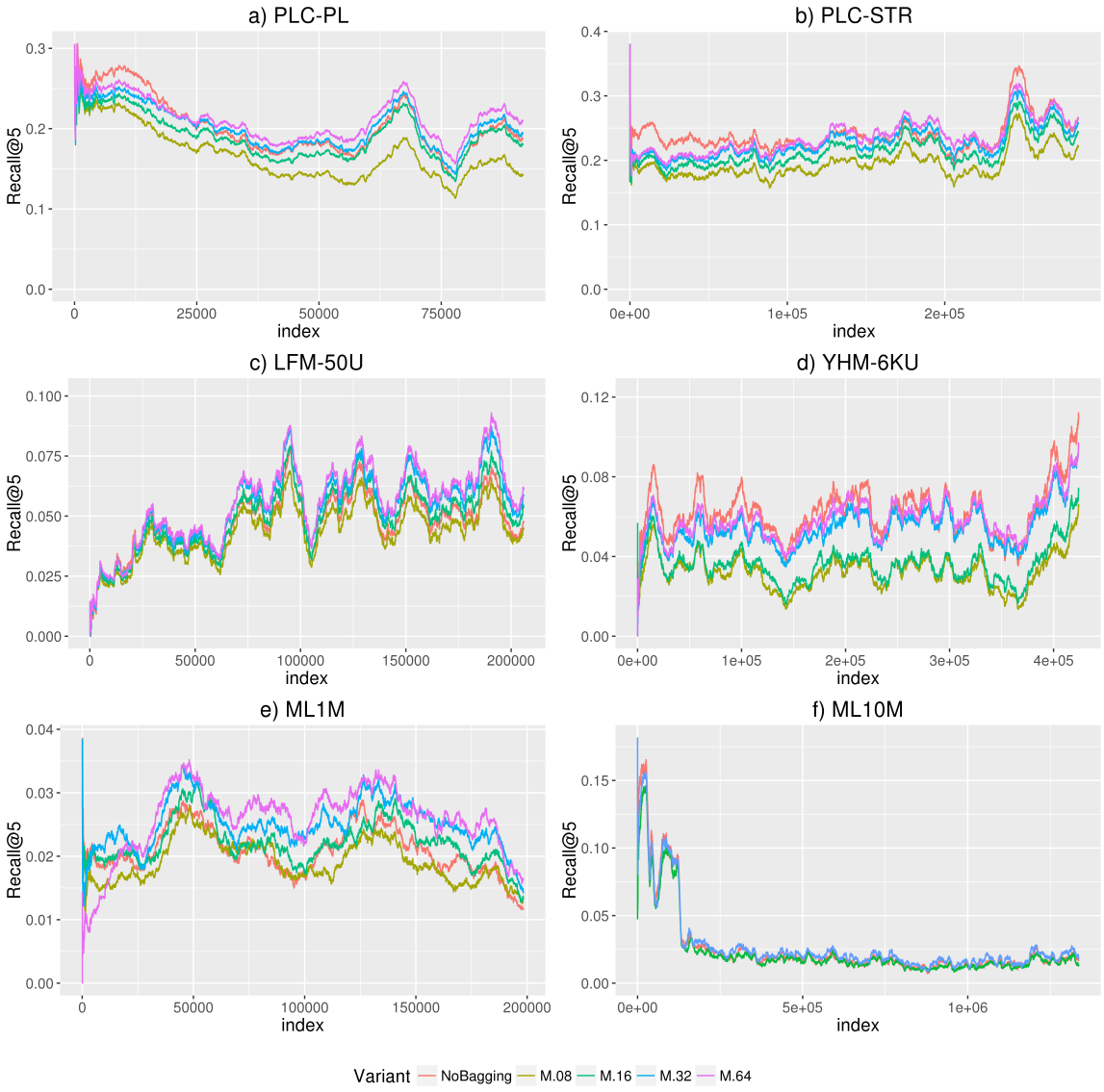


Figure B.2: Prequential evaluation of Recall@5 with ISGD with and without bagging. Lines are drawn using a moving average of Recall@5 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

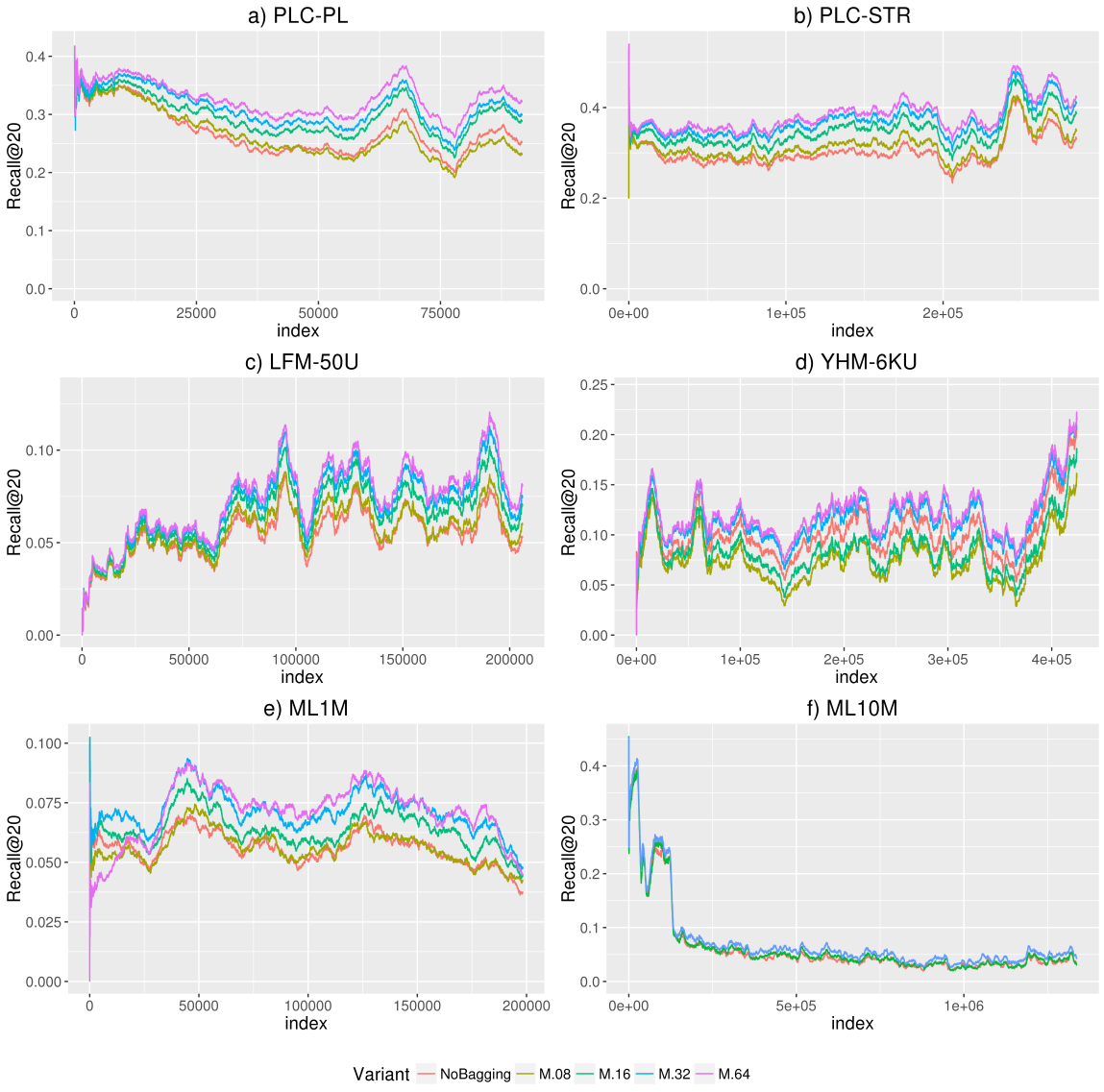


Figure B.3: Prequential evaluation of Recall@20 with ISGD with and without bagging. Lines are drawn using a moving average of Recall@20 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

B.2 ISGD with recency-based negative feedback

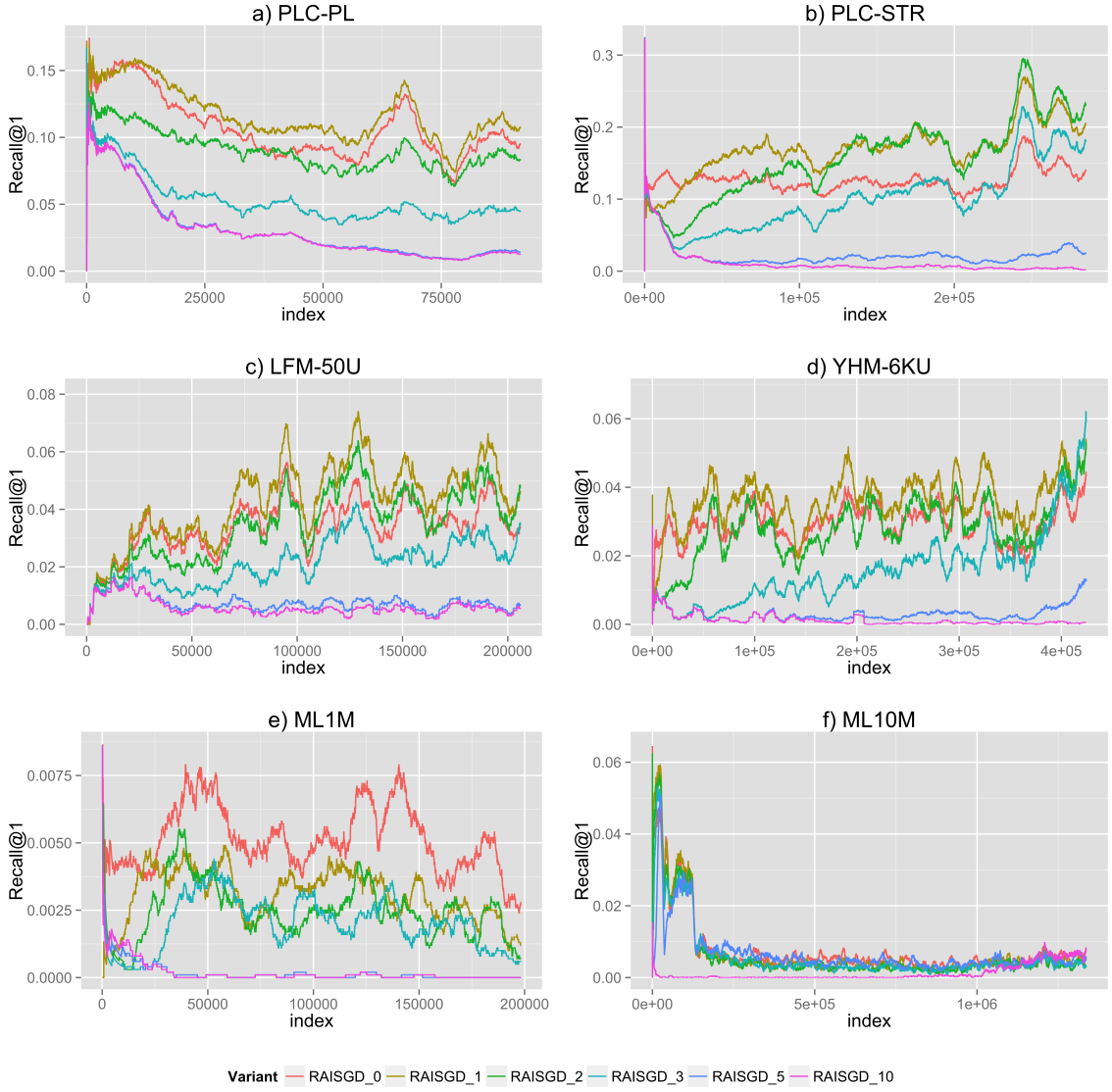


Figure B.4: Prequential evaluation of Recall@1 with 6 datasets for RAISGD with $l \in \{0, 1, 2, 3, 5, 10\}$. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

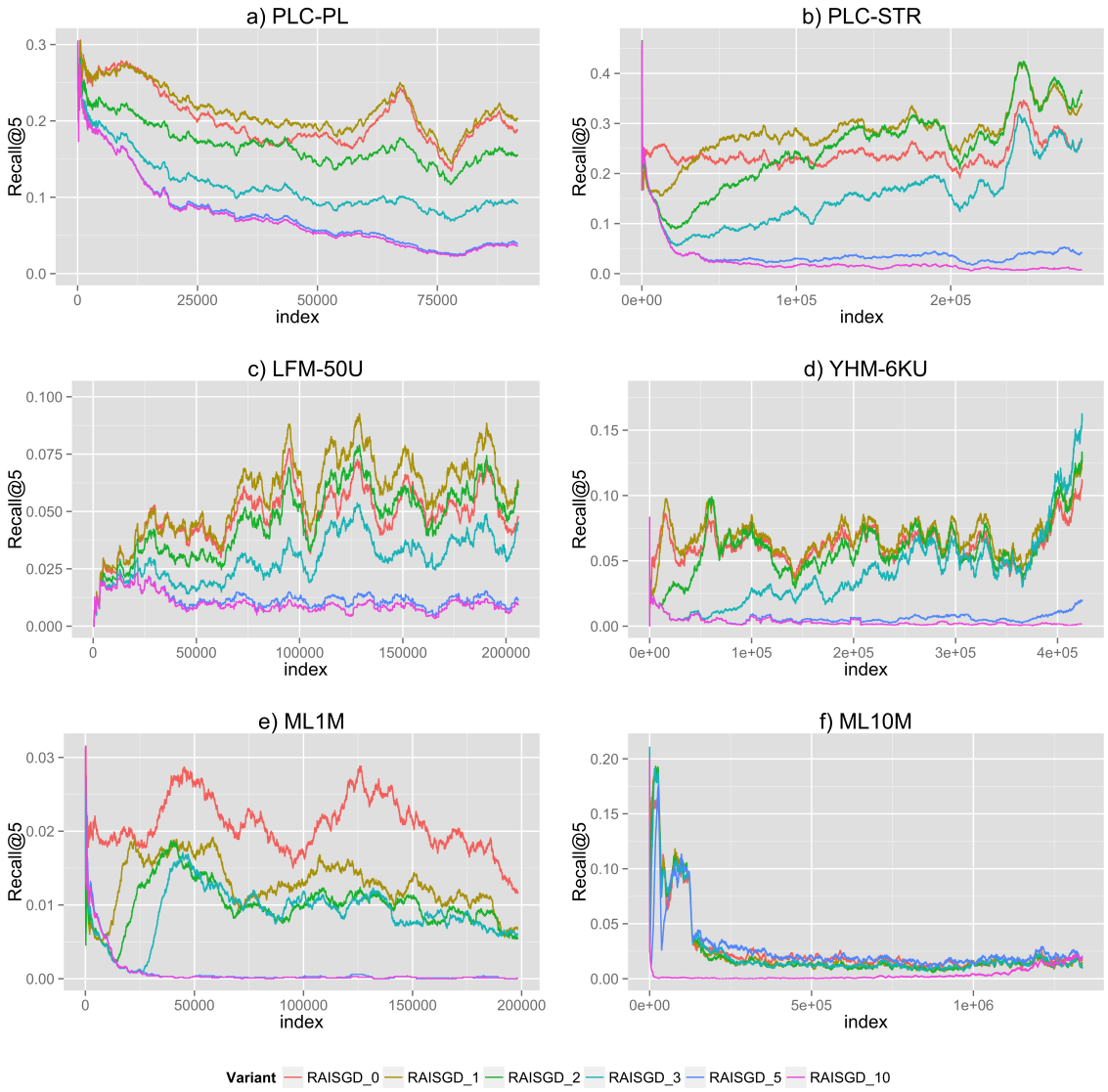


Figure B.5: Prequential evaluation of Recall@5 with 6 datasets for RAISGD with $l \in \{0, 1, 2, 3, 5, 10\}$. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

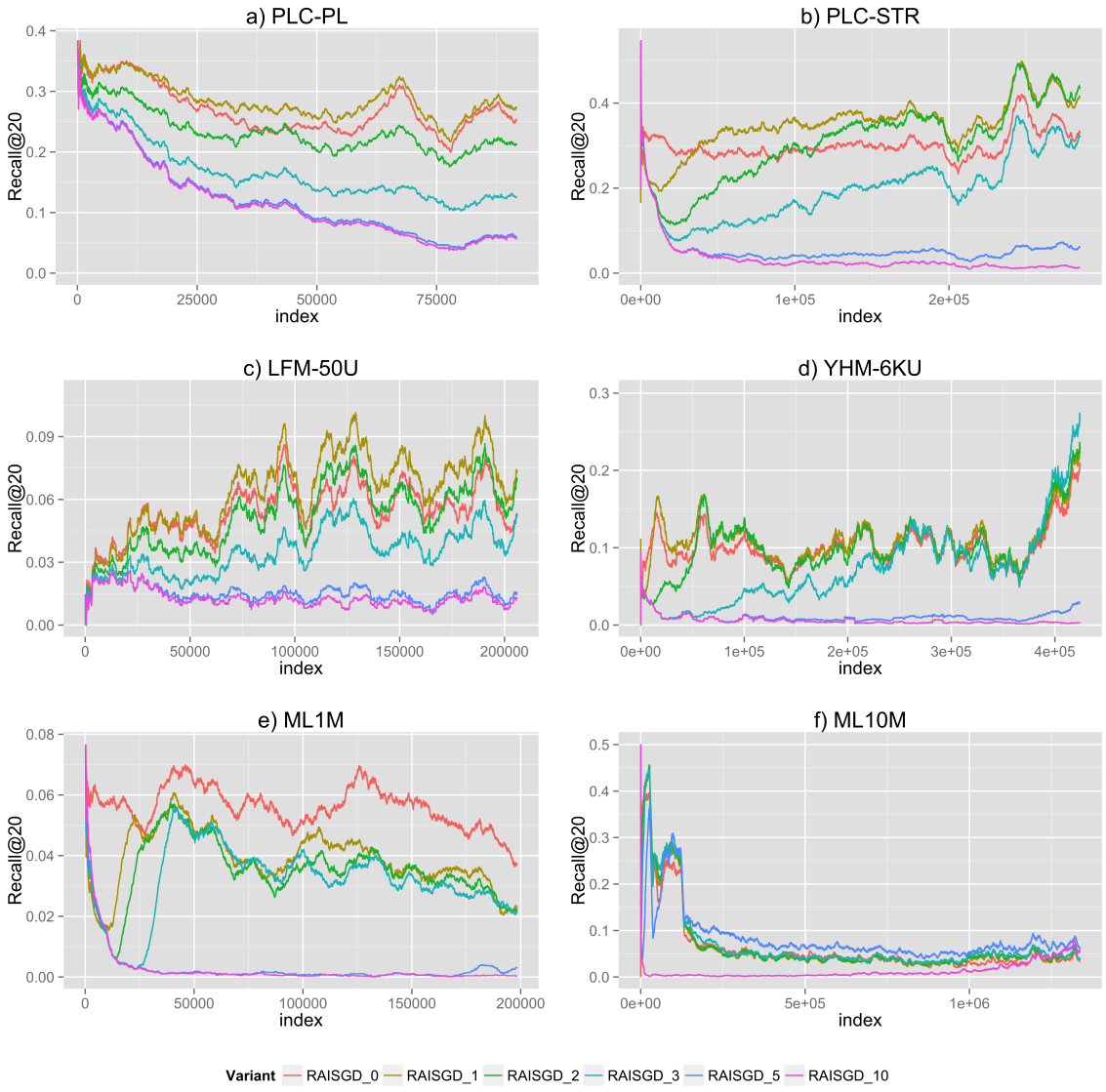


Figure B.6: Prequential evaluation of Recall@20 with 6 datasets for RAISGD with $l \in \{0, 1, 2, 3, 5, 10\}$. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

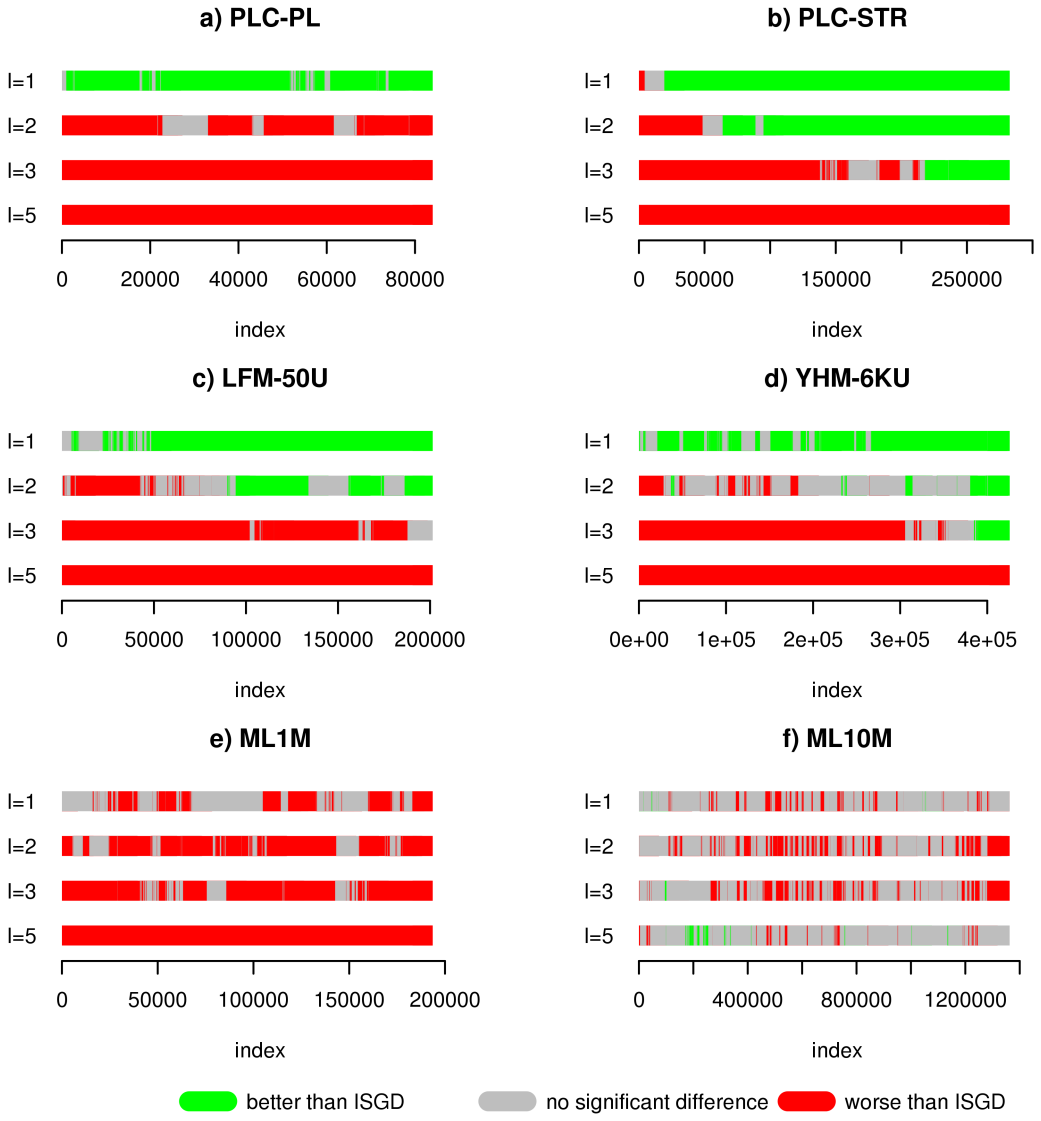


Figure B.7: Signed McNemar pairwise test between Recall@1 obtained by RAISGD with $l \in \{1, 2, 3, 5\}$ with respect to ISGD (RAISGD with $l = 0$). The colour of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that the corresponding value of l is significantly better, without significant difference, or significantly worse than $l = 0$, with a confidence level of 99%.

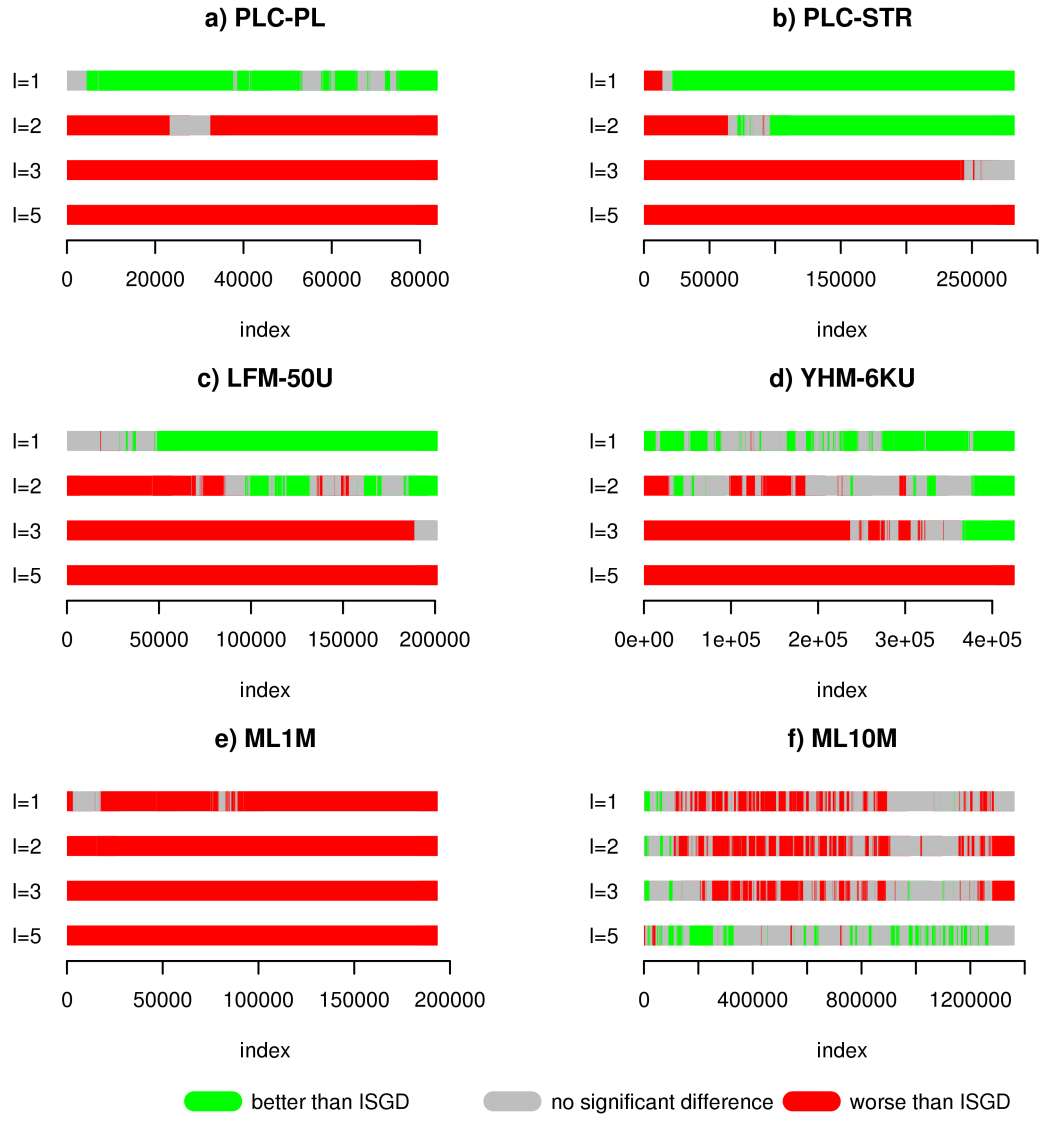


Figure B.8: Signed McNemar pairwise test between Recall@5 obtained by RAISGD with $l \in \{1, 2, 3, 5\}$ with respect to ISGD (RAISGD with $l = 0$). The colour of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that the corresponding value of l is significantly better, without significant difference, or significantly worse than $l = 0$, with a confidence level of 99%.

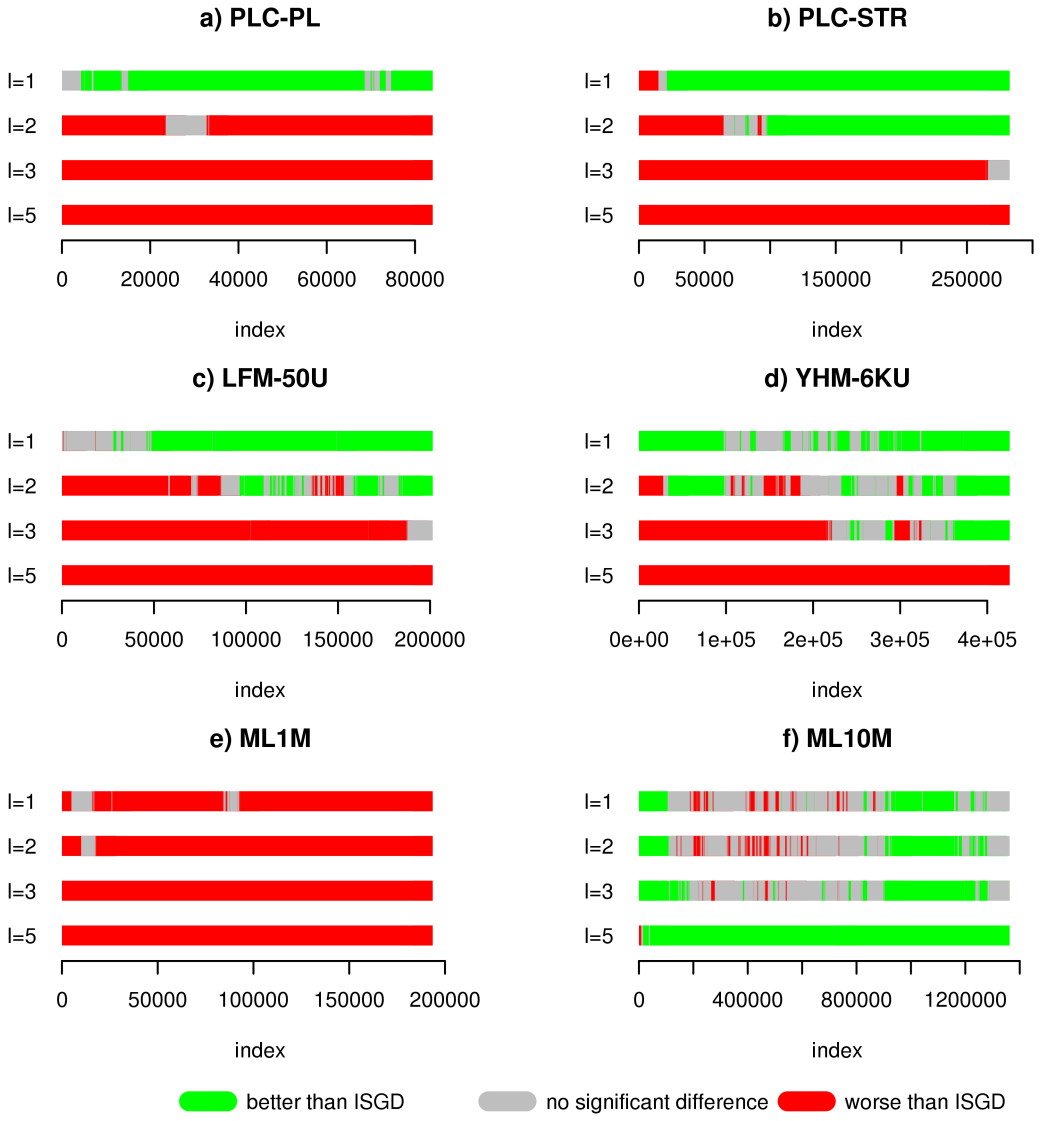


Figure B.9: Signed McNemar pairwise test between Recall@20 obtained by RAISGD with $l \in \{1, 2, 3, 5\}$ with respect to ISGD (RAISGD with $l = 0$). The colour of the bars indicate a value of $M < -6.635$ (red), $-6.635 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that the corresponding value of l is significantly better, without significant difference, or significantly worse than $l = 0$, with a confidence level of 99%.

B.3 Comparison with other algorithms

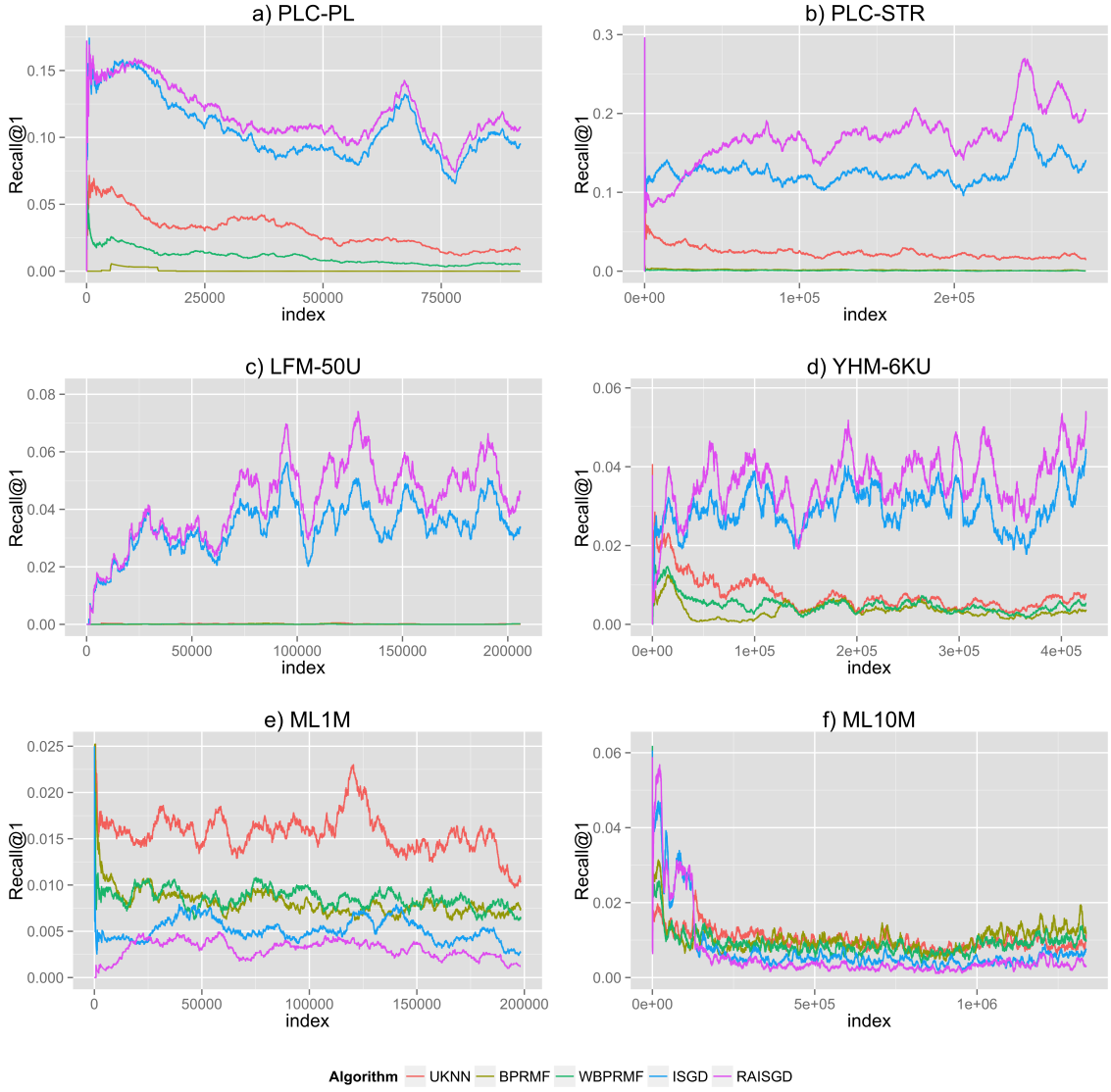


Figure B.10: Prequential evaluation of Recall@1 with 6 datasets. Lines are drawn using a moving average of Recall@1 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

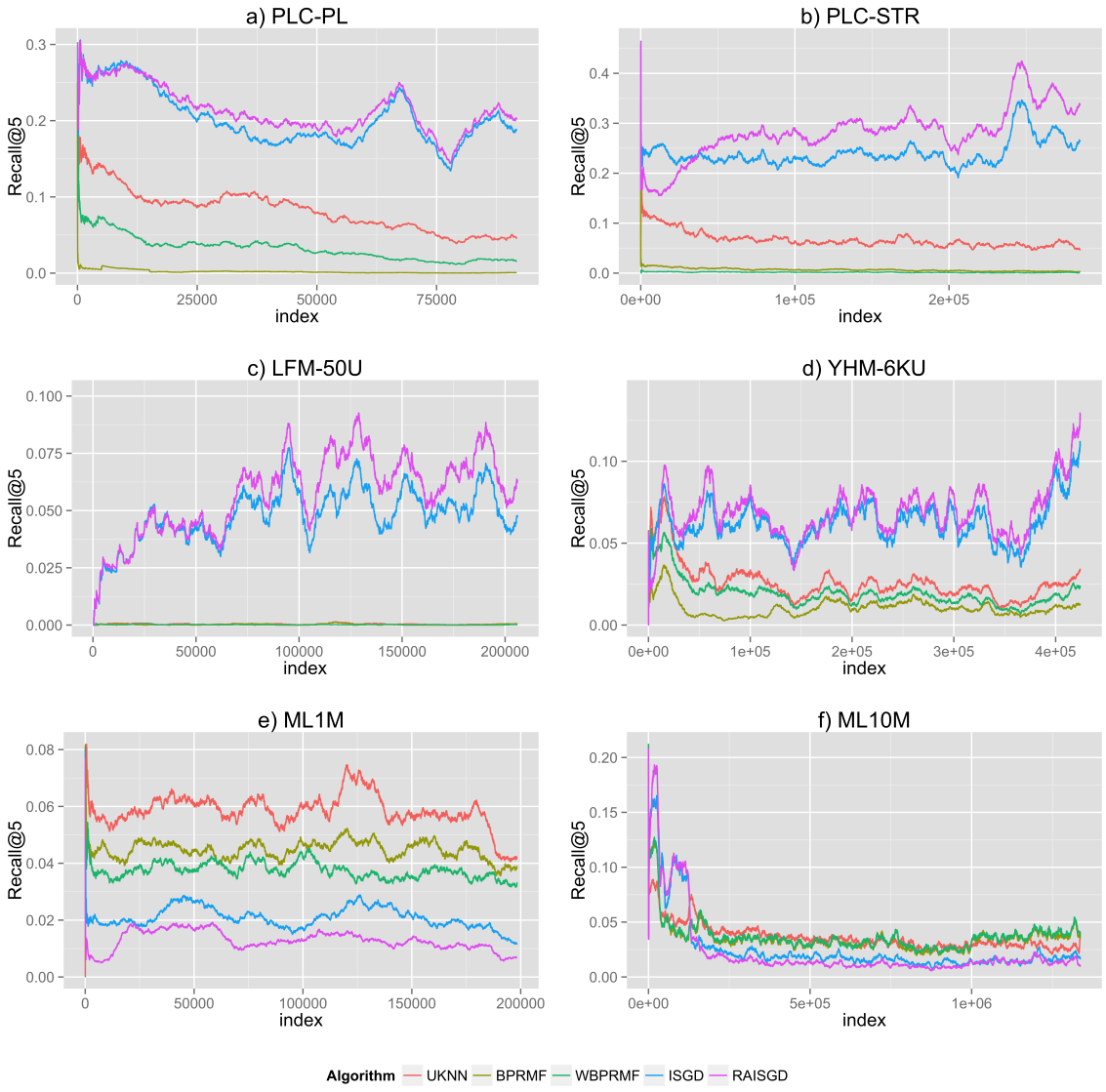


Figure B.11: Prequential evaluation of Recall@5 with 6 datasets. Lines are drawn using a moving average of Recall@5 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

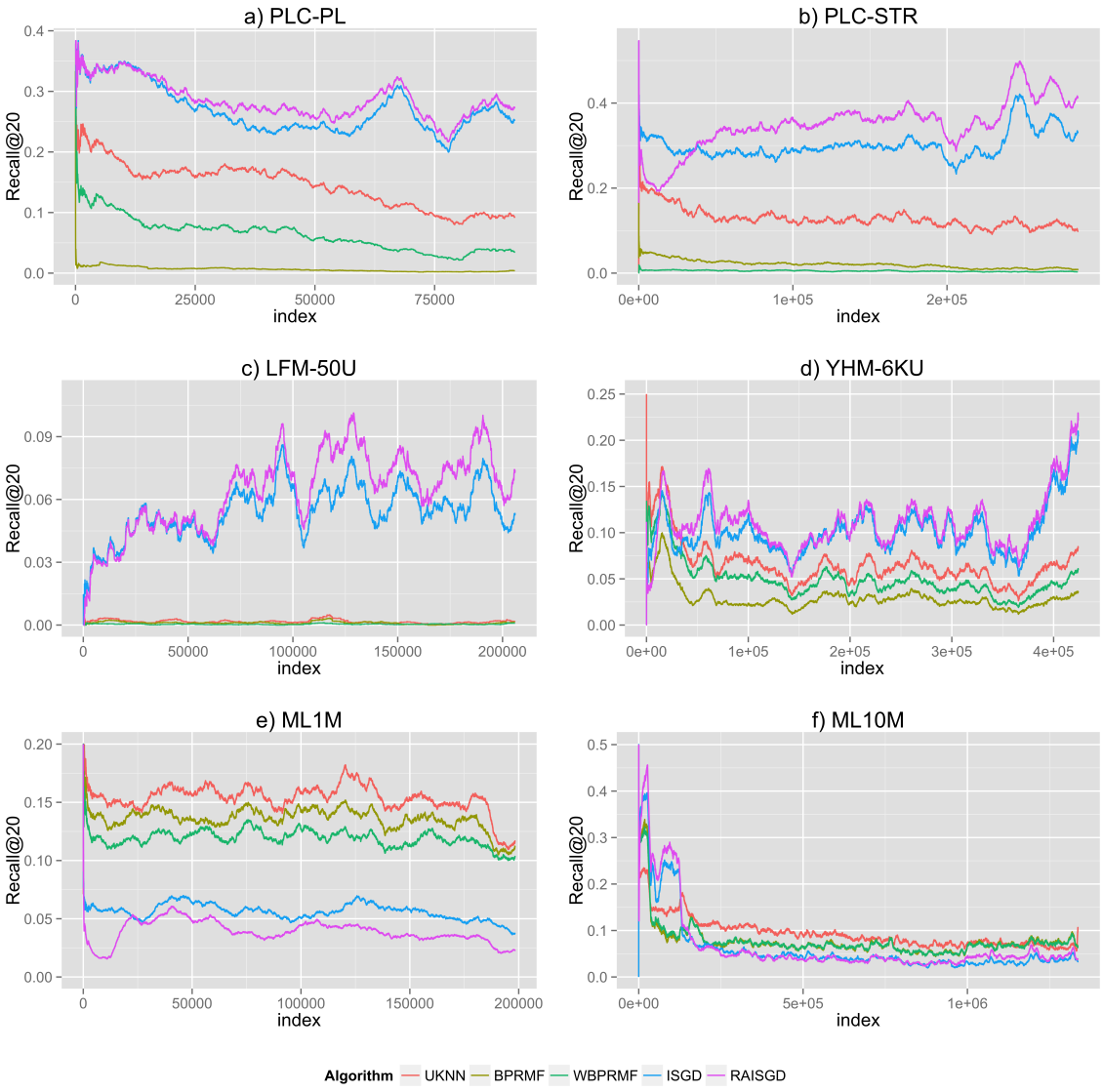


Figure B.12: Prequential evaluation of Recall@20 with 6 datasets. Lines are drawn using a moving average of Recall@10 with $n = 10000$. The first 10 000 points are drawn using the accumulated average.

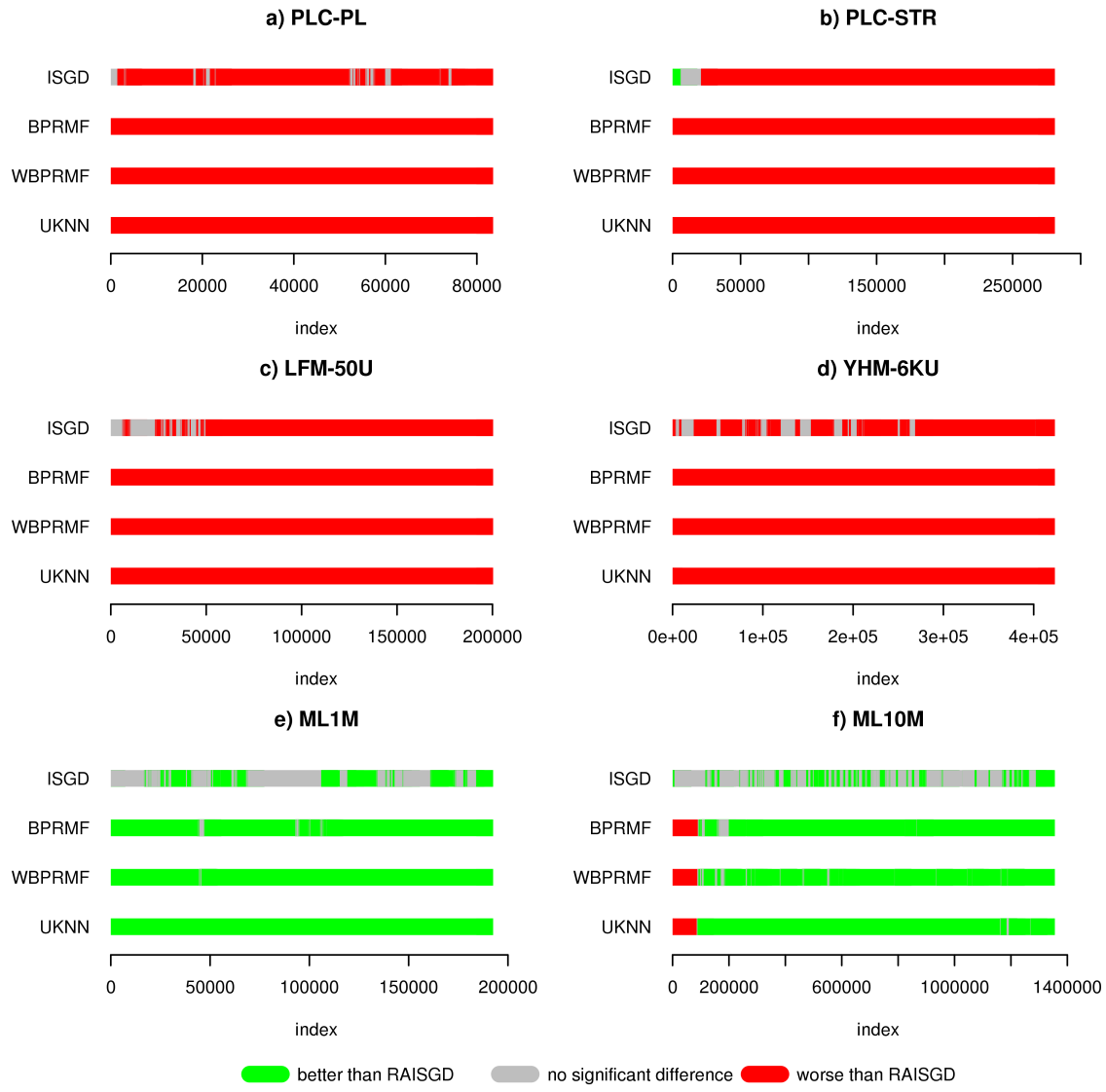


Figure B.13: McNemar pairwise tests between RAISGD and other algorithms, relative to the Recall@1 metric. The colour of the bars indicate a value of $M < -6.635$ (red), $-6.645 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that RAISGD is significantly better, without significant difference, or significantly worse than the referenced algorithm, with a confidence level of 99%.

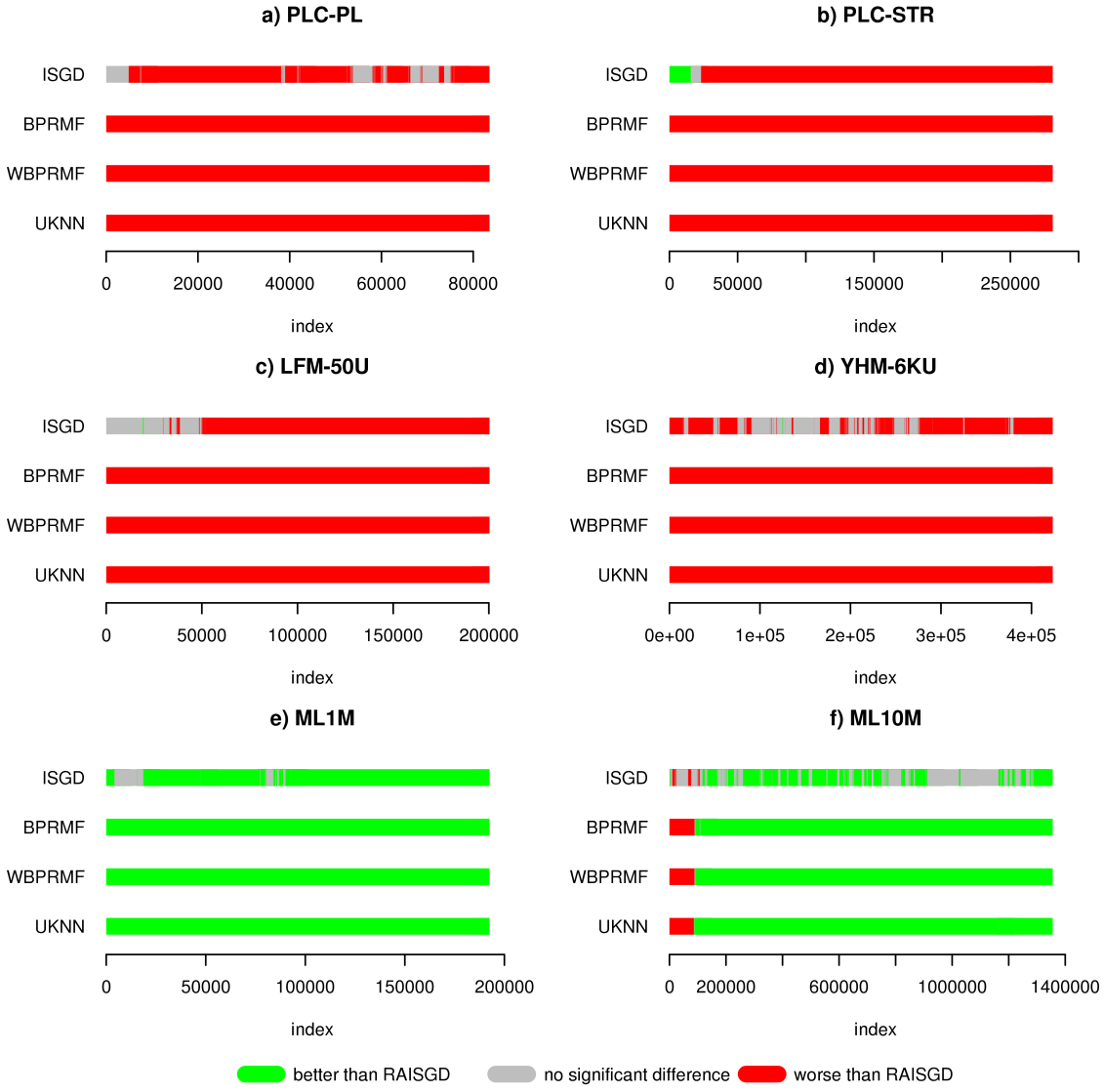


Figure B.14: McNemar pairwise tests between RAISGD and other algorithms, relative to the Recall@5 metric. The colour of the bars indicate a value of $M < -6.635$ (red), $-6.645 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that RAISGD is significantly better, without significant difference, or significantly worse than the referenced algorithm, with a confidence level of 99%.

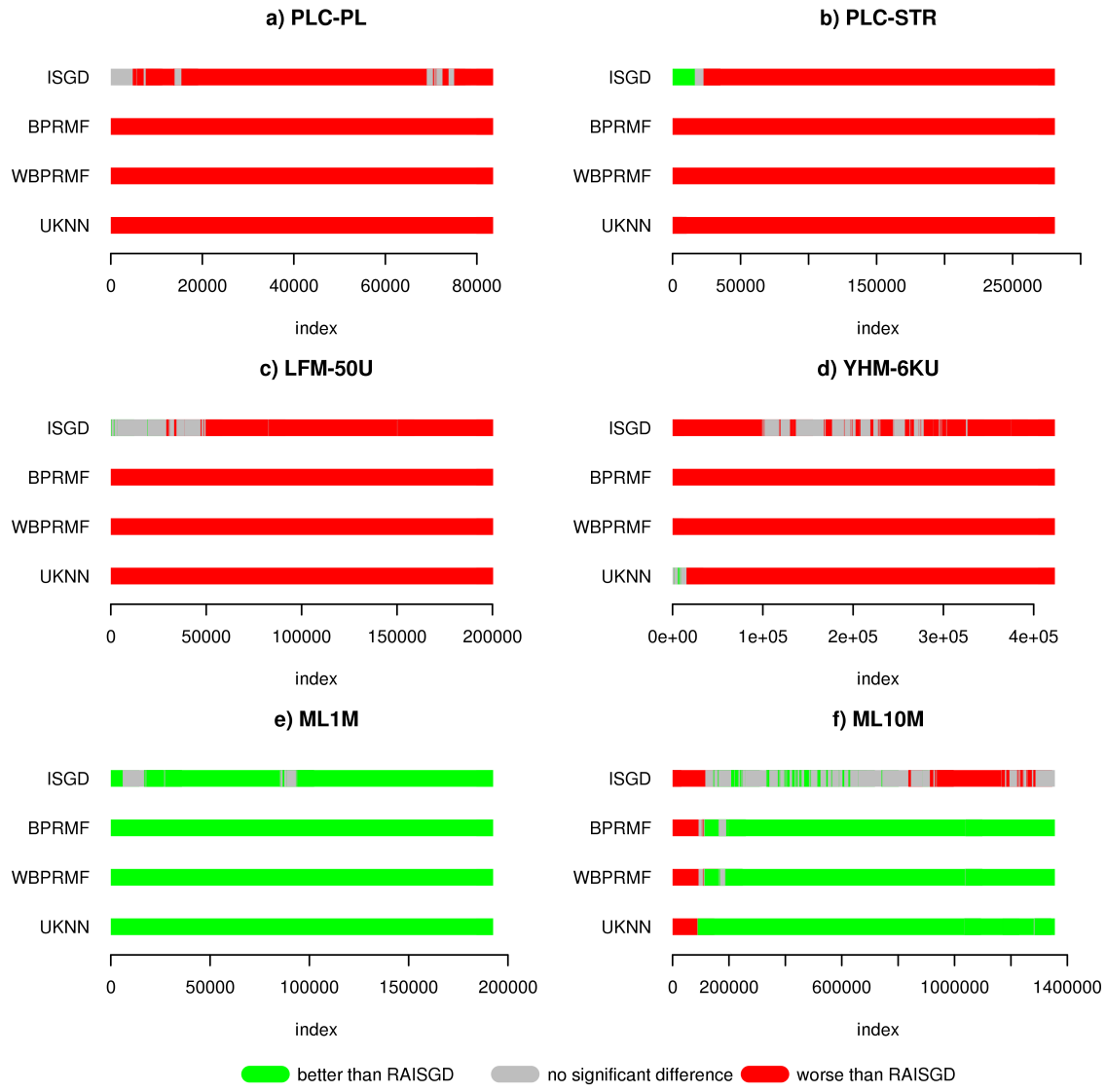


Figure B.15: McNemar pairwise tests between RAISGD and other algorithms, relative to the Recall@20 metric. The colour of the bars indicate a value of $M < -6.635$ (red), $-6.645 \leq M \leq 6.635$ (gray) or $M > 6.635$ (green), indicating that RAISGD is significantly better, without significant difference, or significantly worse than the referenced algorithm, with a confidence level of 99%.

Bibliography

Acar, E., Dunlavy, D. M., Kolda, T. G., and Mørup, M. (2010). Scalable tensor factorizations with missing data. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 701–712. SIAM.

Adomavicius, G., Mobasher, B., Ricci, F., and Tuzhilin, A. (2011). Context-aware recommender systems. *AI Magazine*, 32(3):67–80.

Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A. (2005). Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145.

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749.

Aggarwal, C. C. (2014). A survey of stream classification algorithms. In Aggarwal, C. C., editor, *Data Classification: Algorithms and Applications*, pages 245–274. CRC Press.

Aggarwal, C. C., Wolf, J. L., Wu, K., and Yu, P. S. (1999). Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*, pages 201–212. ACM.

Alexa Website Rankings (2015). `http://www.alexa.com`.

Arel, I., Rose, D., and Karnowski, T. P. (2010). Deep machine learning - A new frontier in artificial intelligence research [research frontier]. *IEEE Comp. Int. Mag.*, 5(4):13–18.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In Popa, L., editor, *PODS '02: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM.

Balabanovic, M. (1997). An adaptive web page recommendation service. In *Proceedings of the First International Conference on Autonomous Agents, Marina del Rey, CA, USA, February 5-8, 1997*, pages 378–385. ACM Press.

Baltrunas, L. and Amatriain, X. (2009). Towards Time-Dependant Recommendation based on Implicit Feedback. In *Workshop on Context-aware Recommender Systems (CARS 2009) at RecSys, New York*.

Bell, R. M. and Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 43–52. IEEE Computer Society.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bennett, J., Lanning, S., and Netflix, N. (2007). The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*.

Bifet, A., Morales, G. D. F., Read, J., Holmes, G., and Pfahringer, B. (2015). Efficient online evaluation of big data stream classifiers. In Cao, L., Zhang, C., Joachims, T., Webb, G. I., Margineantu, D. D., and Williams, G., editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 59–68. ACM.

Billsus, D. and Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconson, USA, July 24-27, 1998*, pages 46–54. Morgan Kaufmann.

Borror, C. M., Champ, C. W., and Rigdon, S. E. (1998). Poisson ewma control charts. *Journal of Quality Technology*, 30(4):352–361.

Bottou, L. (2003). Stochastic learning. In Bousquet, O., von Luxburg, U., and Rätsch, G., editors, *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 146–168. Springer.

Brazdil, P., Giraud-Carrier, C. G., Soares, C., and Vilalta, R. (2009). *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer.

Breese, J. S., Heckerman, D., and Kadie, C. M. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, July 24-26, 1998, University of Wisconsin Business School, Madison, Wisconsin, USA*, pages 43–52. Morgan Kaufmann.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. N. (2005). Learning to rank using gradient descent. In Raedt, L. D. and Wrobel, S., editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 89–96. ACM.

Burke, R. D. (2002). Hybrid recommender systems: Survey and experiments. *User Model. User-Adapt. Interact.*, 12(4):331–370.

Burke, R. D. (2007). Hybrid web recommender systems. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer.

Campos, P. G., Bellogín, A., Díez, F., and Chavarriaga, J. E. (2010). Simple time-biased knn-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 20–23, New York, NY, USA. ACM.

Campos, P. G., Díez, F., and Cantador, I. (2014). Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Model. User-Adapt. Interact.*, 24(1-2):67–119.

Cao, H., Chen, E., Yang, J., and Xiong, H. (2009). Enhancing recommender systems under volatile userinterest drifts. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1257–1266. ACM.

Celma, Ò. (2010). *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer.

Celma, O. (2013). `http://ocelma.net/MusicRecommendationDataset` [accessed jan 2013].

Das, A., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 271–280. ACM.

de Andrade Silva, J., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C. P. L. F., and Gama, J. (2013). Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1):13.

de Castro, P., de Franca, F., Ferreira, H., and Von Zuben, F. (2007). Applying biclustering to perform collaborative filtering. In *Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on*, pages 421–426.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150. USENIX Association.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *JASIS*, 41(6):391–407.

Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., and Nejdl, W. (2012). Real-time top-n recommendation in social streams. In Cunningham, P., Hurley, N. J., Guy, I., and Anand, S. S., editors, *RecSys*, pages 59–66. ACM.

Ding, Y. and Li, X. (2005). Time weight collaborative filtering. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 485–492. ACM.

Ding, Y., Li, X., and Orlowska, M. E. (2006). Recency-based collaborative filtering. In *Database Technologies 2006, Proceedings of the 17th Australasian Database Conference, ADC 2006, Hobart, Tasmania, Australia, January 16-19 2006*, volume 49 of *CRPIT*, pages 99–107. Australian Computer Society.

Domingos, P. and Hulten, G. (2001). Catching up with the data: Research issues in mining data streams. In *DMKD*.

Domingos, P. M. and Hulten, G. (2000). Mining high-speed data streams. In Ramakrishnan, R., Stolfo, S. J., Bayardo, R. J., and Parsa, I., editors, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 71–80. ACM.

Domingues, M. A., Gouyon, F., Jorge, A. M., Leal, J. P., Vinagre, J., Lemos, L., and Sordo, M. (2013). Combining usage and content in an online recommendation system for music in the long tail. *IJMIR*, 2(1):3–13.

Domingues, M. A., Jorge, A. M., and Soares, C. (2008). The impact of contextual information on the accuracy of existing recommender systems for web personalization. In *Web Intelligence*, pages 789–792. IEEE Computer Society.

Dror, G., Koenigstein, N., Koren, Y., and Weimer, M. (2012). The yahoo! music dataset and kdd-cup '11. *Journal of Machine Learning Research - Proceedings Track*, 18:8–18.

Félix, C., Soares, C., Jorge, A., and Vinagre, J. (2014). Monitoring recommender systems: A business intelligence approach. In Murgante, B., Misra, S., Rocha, A. M. A. C., Torre, C. M., Rocha, J. G., Falcão, M. I., Taniar, D., Apduhan, B. O., and Gervasi, O., editors, *Computational Science and Its Applications - ICCSA 2014 - 14th International*

Conference, Guimarães, Portugal, June 30 - July 3, 2014, Proceedings, Part VI, volume 8584 of *Lecture Notes in Computer Science*, pages 277–288. Springer.

Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. (2007). Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowl. Data Eng.*, 19(3):355–369.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In Saitta, L., editor, *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 148–156. Morgan Kaufmann.

Funk, S. (2006). `http://sifter.org/~simon/journal/20061211.html` [Accessed Jan 2013].

Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series. CRC Press.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. P. (2004). Learning with drift detection. In Bazzan, A. L. C. and Labidi, S., editors, *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *KDD*, pages 329–338. ACM.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346.

Gantner, Z., Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2011). Mymedialite: a free recommender system library. In Mobasher, B., Burke, R. D., Jannach, D., and Adomavicius, G., editors, *RecSys*, pages 305–308. ACM.

Gantner, Z., Rendle, S., and Schmidt-Thieme, L. (2010). Factorization models for context-/time-aware movie recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 14–19, New York, NY, USA. ACM.

Gao, H., Tang, J., Hu, X., and Liu, H. (2013). Exploring temporal effects for location recommendation on location-based social networks. In *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 93–100. ACM.

Gemulla, R., Nijkamp, E., Haas, P. J., and Sismanis, Y. (2011). Large-scale matrix factorization with distributed stochastic gradient descent. In Apté, C., Ghosh, J., and Smyth, P., editors, *Proceedings of the 17th ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011, pages 69–77. ACM.

George, T. and Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. In *ICDM 2005: Proceedings of the 5th IEEE International Conference on Data Mining, 27-30 November 2005, Houston, Texas, USA*, pages 625–628. IEEE Computer Society.

Goldberg, K. Y., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151.

Gorgoglione, M. and Panniello, U. (2009). Including context in a transactional recommender system using a pre-filtering approach: Two real e-commerce applications. In *AINA Workshops*, pages 667–672. IEEE Computer Society.

Gori, M. and Pucci, A. (2007). Itemrank: A random-walk based scoring algorithm for recommender engines. In Veloso, M. M., editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2766–2771.

Grouplens (2013). `http://www.grouplens.org/data` [Accessed Jan 2013].

Hand, D. J., Smyth, P., and Mannila, H. (2001). *Principles of data mining*. MIT Press, Cambridge, MA, USA.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.

Hill, W. C., Stead, L., Rosenstein, M., and Furnas, G. W. (1995). Recommending and evaluating choices in a virtual community of use. In *Human Factors in Computing Systems, CHI '95 Conference Proceedings, Denver, Colorado, USA, May 7-11, 1995.*, pages 194–201. ACM/Addison-Wesley.

Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115.

Hong, W., Li, L., and Li, T. (2012). Product recommendation with temporal dynamics. *Expert Syst. Appl.*, 39(16):12398–12406.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272. IEEE Computer Society.

Huang, Z., Chung, W., and Chen, H. (2004). A graph model for e-commerce recommender systems. *JASIST*, 55(3):259–274.

Huang, Z., Chung, W., Ong, T.-H., and Chen, H. (2002). A graph-based recommender system for digital library. In *JCDL*, pages 65–73. ACM.

Hulten, G., Spencer, L., and Domingos, P. M. (2001). Mining time-changing data streams. In Lee, D., Schkolnick, M., Provost, F. J., and Srikant, R., editors, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 97–106. ACM.

Indyk, P. (1999). A small approximately min-wise independent family of hash functions. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland.*, pages 454–456. ACM/SIAM.

Jahrer, M., Töscher, A., and Legenstein, R. A. (2010). Combining predictions for accurate recommender systems. In Rao, B., Krishnapuram, B., Tomkins, A., and Yang, Q., editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 693–702. ACM.

Jannach, D., Lerche, L., and Gdaniec, M. (2013). Re-ranking recommendations based on predicted short-term interests - a protocol and first experiment. In *ITWP 2013: Proceedings of the workshop Intelligent Techniques for Web Personalization and Recommender Systems at AAAI 2013, Bellevue, Washington, 2013*.

Khan, S. S. and Madden, M. G. (2014). One-class classification: taxonomy of study and review of techniques. *Knowledge Eng. Review*, 29(3):345–374.

Khoshneshin, M. and Street, W. N. (2010). Collaborative filtering via euclidean embedding. In *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*, pages 87–94. ACM.

Knijnenburg, B. P., Willemsen, M. C., Gantner, Z., Soncu, H., and Newell, C. (2012). Explaining the user experience of recommender systems. *User Model. User-Adapt. Interact.*, 22(4-5):441–504.

Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data Min. Knowl. Discov.*, 18(1):140–181.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 426–434. ACM.

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 447–456. ACM.

Koychev, I. (2000). Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning*, pages 101–106.

Lathia, N. (2010). *Evaluating collaborative filtering over time*. PhD thesis, University College London.

Lathia, N., Hailes, S., and Capra, L. (2009). Temporal collaborative filtering with adaptive neighbourhoods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 796–797. ACM.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. (1996). Effiicient backprop. In Orr, G. B. and Müller, K., editors, *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer.

Lee, D., Park, S. E., Kahng, M., Lee, S., and goo Lee, S. (2010). Exploiting contextual information from event logs for personalized recommendation. In Lee, R. Y., editor, *Computer and Information Science*, volume 317 of *Studies in Computational Intelligence*, pages 121–139. Springer.

Li, X., Barajas, J. M., and Ding, Y. (2007). Collaborative filtering on streaming data with interest-drifting. *Intell. Data Anal.*, 11(1):75–87.

Lin, W., Alvarez, S. A., and Ruiz, C. (2002). Efficient adaptive-support association rule mining for recommender systems. *Data Min. Knowl. Discov.*, 6(1):83–105.

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

Ling, G., Yang, H., King, I., and Lyu, M. R. (2012). Online learning for collaborative filtering. In *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012*, pages 1–8. IEEE.

Liu, N. N., Cao, B., Zhao, M., and Yang, Q. (2010a). Adapting neighborhood and matrix factorization models for context aware recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 7–13, New York, NY, USA. ACM.

Liu, N. N., He, L., and Zhao, M. (2013). Social temporal collaborative ranking for context aware movie recommendation. *ACM TIST*, 4(1):15.

Liu, N. N., Zhao, M., Xiang, E. W., and Yang, Q. (2010b). Online evolutionary collaborative filtering. In *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*, pages 95–102. ACM.

Liu, T. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Liu, T. (2011). *Learning to Rank for Information Retrieval*. Springer.

Liu, X. and Aberer, K. (2014). Towards a dynamic top-n recommendation framework. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 217–224. ACM.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2012). Distributed graphlab: A framework for machine learning in the cloud. *PVLDB*, 5(8):716–727.

Matuszyk, P. and Spiliopoulou, M. (2014). Selective forgetting for incremental matrix factorization in recommender systems. In Dzeroski, S., Panov, P., Kocev, D., and Todorovski, L., editors, *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8777 of *Lecture Notes in Computer Science*, pages 204–215. Springer.

Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A. M., and Gama, J. (2015). Forgetting methods for incremental matrix factorizatin in recommender systems. In *SAC 2015: Proceedings of the 30th ACM SIGAPP Symposium on Applied Computing, April 13-17, 2015, Salamanca, Spain*, pages 947–953. ACM.

McFee, B., Bertin-Mahieux, T., Ellis, D. P. W., and Lanckriet, G. R. G. (2012). The million song dataset challenge. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 909–916. ACM.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, pages 1097–1101. ACM.

Min, S.-H. and Han, I. (2005). Detection of the customer time-variant pattern for improving recommender systems. *Expert Systems with Applications*, 28(2):189 – 199.

Miranda, C. and Jorge, A. M. (2008). Incremental collaborative filtering for binary ratings. In *2008 IEEE / WIC / ACM International Conference on Web Intelligence, WI 2008, 9-12 December 2008, Sydney, NSW, Australia, Main Conference Proceedings*, pages 389–392. IEEE Computer Society.

Miranda, C. and Jorge, A. M. (2009). Item-based and user-based incremental collaborative filtering for web recommendations. In Lopes, L. S., Lau, N., Mariano, P., and Rocha, L. M., editors, *Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence, EPIA 2009, Aveiro, Portugal, October 12-15, 2009. Proceedings*, volume 5816 of *Lecture Notes in Computer Science*, pages 673–684. Springer.

Mobasher, B., Dai, H., Luo, T., and Nakagawa, M. (2001). Effective personalization based on association rule discovery from web usage data. In *3rd International Workshop on Web Information and Data Management (WIDM 2001), Friday, 9 November 2001, In Conjunction with ACM CIKM 2001, Doubletree Hotel Atlanta-Buckhead, Atlanta, Georgia, USA. ACM, 2001*, pages 9–15. ACM.

Mooney, R. J. and Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries, June 2-7, 2000, San Antonio, TX, USA*, pages 195–204.

Nasraoui, O., Cerwinske, J., Rojas, C., and González, F. A. (2007). Performance of recommendation systems in dynamic streaming environments. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM.

Nasraoui, O., Uribe, C. C., Coronel, C. R., and González, F. A. (2003). Tecno-streams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pages 235–242. IEEE Computer Society.

Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action*. Manning Publications Co., Greenwich, CT, USA.

Oza, N. C. and Russell, S. J. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In Lee, D., Schkolnick, M., Provost, F. J., and Srikant, R., editors, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 359–364. ACM.

Pálovics, R., Benczúr, A. A., Kocsis, L., Kiss, T., and Frigó, E. (2014). Exploiting temporal influence in online recommendation. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 273–280. ACM.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R. M., Scholz, M., and Yang, Q. (2008). One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on*

Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy, pages 502–511. IEEE Computer Society.

Panniello, U., Tuzhilin, A., and Gorgoglione, M. (2014). Comparing context-aware recommender systems in terms of accuracy and diversity. *User Model. User-Adapt. Interact.*, 24(1-2):35–65.

Panniello, U., Tuzhilin, A., Gorgoglione, M., Palmisano, C., and Pedone, A. (2009). Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the 2009 ACM Conference on Recommender Systems, RecSys 2009, New York, NY, USA, October 23-25, 2009*, pages 265–268. ACM.

Papagelis, M., Rousidis, I., Plexousakis, D., and Theoharopoulos, E. (2005). Incremental collaborative filtering for highly-scalable recommendation algorithms. In Hacid, M., Murray, N. V., Ras, Z. W., and Tsumoto, S., editors, *Foundations of Intelligent Systems, 15th International Symposium, ISMIS 2005, Saratoga Springs, NY, USA, May 25-28, 2005, Proceedings*, volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer.

Paquet, U. and Koenigstein, N. (2013). One-class collaborative filtering with random graphs. In Schwabe, D., Almeida, V. A. F., Glaser, H., Baeza-Yates, R. A., and Moon, S. B., editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 999–1008. International World Wide Web Conferences Steering Committee / ACM.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, volume 2007, pages 5–8.

Pazzani, M. J. and Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331.

Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer.

Pu, P., Chen, L., and Hu, R. (2012). Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Model. User-Adapt. Interact.*, 22(4-5):317–355.

Rafailidis, D. and Nanopoulos, A. (2014). Modeling the dynamics of user preferences in coupled tensor factorization. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 321–324. ACM.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). BPR: bayesian personalized ranking from implicit feedback. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461. AUAI Press.

Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 811–820. ACM.

Rendle, S. and Schmidt-Thieme, L. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*, pages 81–90. ACM.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, October 22-26, 1994, Chapel Hill, NC, USA*, pages 175–186.

Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., and Nones, M. (2003). Product recommendation with interactive query management and twofold similarity. In *Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR 2003, Trondheim, Norway, June 23-26, 2003, Proceedings*, volume 2689 of *Lecture Notes in Computer Science*, pages 479–493. Springer.

Rodrigues, A. V., Jorge, A., and Dutra, I. (2015). Accelerating recommender systems using gpus. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 879–884. ACM.

Said, A. and Bellogín, A. (2014). Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 129–136. ACM.

Said, A., Berkovsky, S., and De Luca, E. W. (2010). Putting things in context: Challenge on context-aware movie recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 2–6, New York, NY, USA. ACM.

Salakhutdinov, R. and Mnih, A. (2007). Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1257–1264. Curran Associates, Inc.

Salakhutdinov, R. and Mnih, A. (2008). Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 880–887. ACM.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000a). Analysis of recommendation algorithms for e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–167.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295. ACM.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2000b). Application of dimensionality reduction in recommender system - a case study. In *WEBKDD'2000: Web Mining for E-Commerce – Challenges and Opportunities August 20, 2000, Boston, MA*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Segrera, S. and Moreno, M. N. (2006). An experimental comparative study of web mining methods for recommender systems. In *Proceedings of the 6th WSEAS International Conference on Distance Learning and Web Engineering*, pages 56–61. World Scientific and Engineering Academy and Society (WSEAS).

Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 257–297. Springer.

Shani, G., Heckerman, D., and Brafman, R. I. (2005). An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth". In *Human Factors in Computing Systems, CHI '95 Conference Proceedings, Denver, Colorado, USA, May 7-11, 1995.*, pages 210–217. ACM/Addison-Wesley.

Shi, Y., Larson, M., and Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1):3.

Siddiqui, Z. F., Tiakas, E., Symeonidis, P., Spiliopoulou, M., and Manolopoulos, Y. (2014). xstreams: Recommending items to users with time-evolving preferences. In Akerkar, R., Bassiliades, N., Davies, J., and Ermolayev, V., editors, *WIMS*, page 22. ACM.

Similarweb Website Rankings (2015). http://www.similarweb.com.

Smola, A. J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 911–918. Morgan Kaufmann.

Symeonidis, P., Nanopoulos, A., Papadopoulos, A., and Manolopoulos, Y. (2006). Nearest-biclusters collaborative filtering with constant values. In *Advances in Web Mining and Web Usage Analysis, 8th International Workshop on Knowledge Discovery on the Web, WebKDD 2006, Philadelphia, PA, USA, August 20, 2006, Revised Papers*, volume 4811 of *Lecture Notes in Computer Science*, pages 36–55. Springer.

Takacs, G., Pilaszy, I., Nemeth, B., and Tikk, D. (2007). On the gravity recommendation system. In *Proceedings of KDD Cup and Workshop*, volume 2007.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656.

Tiroshi, A., Berkovsky, S., Kâafar, M. A., Vallet, D., and Kuflik, T. (2014). Graph-based recommendations: Make the most out of social data. In Dimitrova, V., Kuflik, T., Chin, D., Ricci, F., Dolog, P., and Houben, G., editors, *User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings*, volume 8538 of *Lecture Notes in Computer Science*, pages 447–458. Springer.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2014). Algorithm selection on data streams. In Dzeroski, S., Panov, P., Kocev, D., and Todorovski, L., editors, *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8777 of *Lecture Notes in Computer Science*, pages 325–336. Springer.

Vinagre, J. and Jorge, A. M. (2012). Forgetting mechanisms for scalable collaborative filtering. *J. Braz. Comp. Soc.*, 18(4):271–282.

Vinagre, J., Jorge, A. M., and Gama, J. (2014a). Evaluation of recommender systems in streaming environments. In *Proceedings of the Workshop on Recommender Systems Evaluation: Dimensions and Design in conjunction with the 8th ACM Conference on Recommender Systems (RecSys 2014), Foster City, CA, USA, October 10, 2014.*, pages 1–6.

Vinagre, J., Jorge, A. M., and Gama, J. (2014b). Fast incremental matrix factorization for recommendation with positive-only feedback. In Dimitrova, V., Kuflik, T., Chin, D., Ricci,

F., Dolog, P., and Houben, G., editors, *User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings*, volume 8538 of *Lecture Notes in Computer Science*, pages 459–470. Springer.

Vinagre, J., Jorge, A. M., and Gama, J. (2015a). Collaborative filtering with recency-based negative feedback. In *SAC 2015: Proceedings of the 30th ACM SIGAPP Symposium on Applied Computing, April 13-17, 2015, Salamanca, Spain*, pages 963–965. ACM.

Vinagre, J., Jorge, A. M., and Gama, J. (2015b). An overview on the exploitation of time in collaborative filtering. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 5(5):195–215.

Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., and Sun, J. (2010). Temporal recommendation on graphs via long- and short-term preference fusion. In Rao, B., Krishnapuram, B., Tomkins, A., and Yang, Q., editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 723–732. ACM.

Xiong, L., Chen, X., Huang, T., Schneider, J. G., and Carbonell, J. G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 211–222. SIAM.

Yahoo (2013). `https://webscope.sandbox.yahoo.com/catalog.php?datatype=r` [accessed jan 2013].

Yin, L., Wang, Y., and Yu, Y. (2012). Collaborative filtering via temporal euclidean embedding. In *Web Technologies and Applications - 14th Asia-Pacific Web Conference, APWeb 2012, Kunming, China, April 11-13, 2012. Proceedings*, volume 7235 of *Lecture Notes in Computer Science*, pages 513–520. Springer.

Yu, H., Hsieh, C., Si, S., and Dhillon, I. S. (2014). Parallel matrix factorization for recommender systems. *Knowl. Inf. Syst.*, 41(3):793–819.

Yuan, Q., Cong, G., Ma, Z., Sun, A., and Magnenat-Thalmann, N. (2013). Time-aware point-of-interest recommendation. In *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 363–372. ACM.

Zimdars, A., Chickering, D. M., and Meek, C. (2001). Using temporal data for making recommendations. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*, pages 580–588. Morgan Kaufmann.